SHI(EL)DS:
A Novel Hardware-based Security Backplane to Enhance Security with Minimal Impact to System Operation

THESIS

Matthew Gundry Judge, Captain, USAF

AFIT/GCE/ENG/08-07

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/GCE/ENG/08-07

SHI(EL)DS:
A Novel Hardware-based Security Backplane to Enhance Security with Minimal Impact to System Operation

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Engineering

Matthew Gundry Judge, B.S.E.E.

Captain, USAF

March 2008

AFIT/GCE/ENG/08-07

SHI(EL)DS:
A Novel Hardware-based Security Backplane to Enhance Security with Minimal Impact to System Operation

Matthew Gundry Judge, B.S.E.E.
Captain, USAF

Approved:

/signed/	11 Mar 2008
Maj P.D. Williams, PhD (Chairman)	date
/signed/	11 Mar 2008
Dr. Y. Kim (Member)	date
/signed/	11 Mar 2008
Dr. B. Mullins (Member)	date

AFIT/GCE/ENG/08-07

Abstract

Computer security continues to increase in importance both in the commercial world and within the Air Force. Dedicated hardware for security purposes presents and enhances a number of security capabilities. Hardware enhances both the security of the *security system* and the quality and trustworthiness of the information being gathered by the security monitors. Hardware reduces avenues of attack on the security system and ensures the trustworthiness of information only through proper design and placement. Without careful system design, security hardware leaves itself vulnerable to many attacks that it is capable of defending against. Our SHI(EL)DS architecture combines these insights into a comprehensive, modular hardware security backplane architecture. This architecture provides many of the capabilities required by the Cybercraft deployment platform. Most importantly, it makes significant progress towards establishing a root of trust for this platform. Progressing the development of the Cybercraft initiative advances the capabilities of the Air Force's ability to operate in and defend cyberspace.

Acknowledgements

First, I would like to thank my advisor, Major Paul Williams, for helping me develop the concepts and information contained within this document and helping to guide me forward to accomplish this research. I would also like to thank Dr. Yong Kim and Dr. Barry Mullins for their input. Finally I would like to thank my colleagues, friends, family, and most importantly my girlfriend for their support and friendship throughout this process.

Matthew Gundry Judge

Table of Contents

Page

Page

List of Figures

List of Tables

List of Abbreviations

Abbreviation		Page
CDIS	Computer Defense Immune System	47
AIS	Artificial Immune System	47
MOEA	Multi-Objective Evolutionary Algorithm	47
jREMISA	java-based Retrovirus Evolutionary Multiobjective Immune System Algorithm	47
VMM	Virtual Machine Monitor	48
VM	Virtual Machine	48
HAV	Hardware Assisted Virtualization	48
TVMM	Trusted Virtual Machine Monitor	49
VMBR	Virtual-Machine Based Rootkits	51
HAV	Hardware Assisted Virtualization	52
SVM	Secure Virtual Machine	52
PCI	Peripheral Component Interface	55
APHID	Anomaly Processor in Hardware for Intrusion Detection	56
DDoS	Distributed Denial of Service	57
CuPIDS	CoProcesso Intrusion Detection System	57
UMA	Uniform Memory Access	57
SPCM	Security Policy Compliance Monitoring	57
CPP	CuPIDS Production Process	57
CSP	CuPIDS Shadow Process	57
SECM	Security Enhanced Chip Multiprocessor	59
L2	Level 2	59
TCAM	Ternary Content Addressable Memory	60
BV	Bit Vector	60
Gbps	Gigabit per Second	60
NIDS	Network Intrusion Detection System	60
CAM	Content Addressable Memory	60
NFA	Nondeterministic Finite Automata	61

Abbreviation		Page
DFA	Deterministic Finite Automata	61
PPU	Production Processor Unit	61
SMU	Shadow Monitoring Unit	61
SRAS	Secure Return Address Stack	63
ISA	Instruction Set Architecture	64
FIFO	First In First Out	64
PC	Program Counter	65
IR	Instruction Register	65
TPM	Trusted Platform Module	67
IPS	Intrustion Prevention System	68
HT	Hyper Transport	73
NUMA	Non-uniform Memory Access	75
PID	Process Identifier	76
GPU	Graphics Processing Unit	79
MMI/O	Memory-mapped Input/Output	79
SNIC	Security Network Interface Card	81
DMA	Direct Memory Access	83
IP	Internet Protocol	95
C3	Command, Control, and Communications	96

SHI(EL)DS:
A Novel Hardware-based Security Backplane to Enhance Security with Minimal Impact to System Operation

I. Introduction

Dedicated hardware provides significant improvement to security solutions when properly designed and implemented. This work explores how and what advantages are gained by a security system through its use. It develop a general security architecture called Secure Hardware Intrusion (Elimination, Limitation, and) Detection System (SHI(EL)DS), which incorporates hardware to improve a system's overall security and provide a basis for the Cybercraft deployment platform. This work supports the Air Force's expanded mission of defending Cyberspace. By providing improved security of our computer systems and supporting the development of Cybercraft, this research helps to protect vital information and assets as more and more of our military becomes reliant on computer systems. This research provides direct application to server networks and critically exposed systems in our networks.

1.1 Background and Problem Overview

Despite computer security continuously increasing importance in today's connected world, the capabilities and speed aspects of computer performance continue to dominate designer's primary goals when creating new systems. With security not

receiving the main focus of designers, the responsibility is pushed from hardware to software developers to implement good programming practices and adapt how an operating system (OS) handles the processes it is given to control. These practices are often ignored, compounding the inherent vulnerabilities of a computer system, and even when considered become increasingly difficult to achieve as systems become more and more complex. A number of hardware-based solutions have been conceived in recent years, though again most are not designed by the primary designers, but left to third party developers and built as peripherals to the system. Although developers such as Intel®have begun to design Trusted Execution Technology into their architectures, they still have limited scope and usability [31]. Despite this work and the ever increasing number of security focused publications, the number of vulnerabilities reported each year has increased 35-fold from 1995 to 2005 and continues to increase through the present [12].

Compounding the issue of creating a viable security solution is the inherent inverse relationship, especially in software based solutions, between how secure a system is and its usability/performance. Few designers and developers are willing to trade performance for security, creating a demand for any security system to provide a significant increase in security for any small amount of performance degradation. Developing a hardware-based security backplane eliminates contention for system resources and leaves a much smaller degradation footprint on a production system, especially in the realm of monitoring.

The current field of computer security creates defenses which are significantly flawed. Lack of realtime monitoring capability and known accurate information severely hamper current security solutions. Once the system software core, the kernel, has been compromised current software security solutions are also compromised since they run as a process managed by the kernel. This problem extends to all software solutions, regardless of their own security measures.

1.2 Research Goals

This research aims to accomplish a number of goals. It attempts to understand the role of hardware for enhancing security within a system. This research explores if it is necessarily or helpful, and how to implement it to achieve improved security. It also evaluates a number of proposed hardware security solutions based on this understanding. This research aims to clearly identify how to ensure security monitor receives accurate data. Without accurate data, any attempts to interpret this data and act on what it is telling the security system runs a much higher risk of providing erroneous operation of the security system.This research explores the security of the information being passed from the production system to the security system and how to categorize it.

Having explored the need for hardware in security, this research leverages an understanding of the role of hardware in security and how to ensure a monitor receives accurate data to develop a dedicated security hardware backplane. The design of this security backplane aims to provide a system with three key advantages. It provides an

unobtrusive design with little or no change to existing production system operation and performance. It also provides maintainable self security even when the security of the production system is compromised. Finally, it leverages hardware primitives identified to provide and enhance unique aspects of monitoring and response. After developing the hardware security backplane architecture, this research presents this architecture as a basis for developing the Cybercraft deployment platform. It attempts to design the security backplane to meet the goals of the Cybercraft initiative.

1.3 Contributions

Through the course of this research, a number of contributions to the field of computer security are presented. They are listed here:

- This research identifies a new axis of security for the information being passed to the security system/monitor from the production system and developed a categorization for the *Trustworthiness of Information* being retrieved by the security system, identifying whether the possibility for compromise of this information is possible.
- This research develops a critical understanding of the need for dedicated hardware to enhance security. It analyzes why software is incapable of securely protecting software when operating on the same system and shows that hardware in and of itself does not necessarily improve on software-based solutions without proper design and location. It also presents advantages gained and en-

hanced through the use of dedicated hardware and explicitly defines hardware requirements to solve inadequacies of software solutions.

- This work develops a novel hardware security backplane architecture that provides a framework for development of security solutions. It utilizes an understanding of the *Trustworthiness of Information* and the requirements for providing improvements through dedicated hardware in security.
- Lastly, this research augments the Cybercraft initiative development by enabling a root of trust for the Cybercraft deployment platform with the backplane architecture. It also provides additional information and capabilities to be utilized by Cybercraft sensor, decision engine, and effector payloads.

1.4 Document Layout

Chapter II provides an overview of different classifications and taxonomies relating to intrusions, intrusion detection, security, and trust. This work provides a basis for a discussion on dedicated hardware for security in Chapter III. It presents research into why dedicated hardware is needed, the advantages which can be gained from it and what precisely is required to realize these advantages. This research also explains what is meant by hardware-based security. To aid in the discussion of the necessity of hardware we develop a classification for the trustworthiness of information, showing that only hardware is capable of providing *First-hand Information*, a necessary condition for guaranteeing accurate monitoring.

With this understanding of hardware this research presents an overview of current security solutions in Chapter IV. It looks at a number of software and virtual machine monitor (VMM) solutions and discuss their weaknesses in terms of the understanding gained from the previous chapter. This work explores a wide range of research presenting hardware solutions in both host and network intrusion detection. It evaluates these hardware solutions in terms of how effective they are in achieving the advantages of hardware laid out in Chapter III and how closely they adhere to the requirements presented there.

After exploring different proposed and implemented security solutions, this research develops a security backplane to provide a comprehensive security solution. Chapter V lays out the SHI(EL)DS architecture. This architecture attempts to achieve unique and enhance current security capabilities, through meeting the requirements outlined in Chapter III. By forming a security backplane concept designed to be modular and minimize modification to the production system, this work presents an architecture that achieves enhanced security without modifying the instruction set architecture (ISA) of the production system. Chapter VI presents the SHI(EL)DS architecture as the basis for creating the Cybercraft deployment platform. Chapter VII concludes this work with a summary of the key contributions and a discussion of future research avenues that are opened by this research.

II. Review of Computer Security Taxonomies and Classifications

Here we present research related to our own work. To frame the discussion of our research we will present work into a number of different topics. First we will look at classifying different types of attacks and methods of intrusion detection (ID). We will present different intrusion detection systems (IDS) which have been implemented along with a number of hardware-based security mechanisms which have been proposed.

2.1 Attacks, Vulnerabilities, and Exploits

There are various types of attack. Viruses, Rootkits, Timing-based Attacks, and Relocation Attacks are described here to aid in the discussion of this work. A limited number of specific exploits are described, which are important motivation to this research and examples of important limitations of current software or hardware solutions.

2.1.1 Classifications of Attack. Numerous works have presented work on creating taxonomies of security flaws [37] and types of attacks [41, 40, 51]. Most classifications of intrusion are developed from the attackers point of view [1, 44]. A few basic descriptions of different types of attacks are presented here to help frame the discussion of Intrusion Detection as compiled by Hart [26] and Mott [48].

Virus A virus is malicious code that is attached to other software. It does not self-propagate.

Worm A worm is malicious code that is self-propagating.

Trojan A trojan is malicious code embedded in innocent software to provide new avenues of attack into a system. Can be either non-self-propagating or self-propagating.

Rootkit A rootkit is code that relies on root level access to modify system call interaction.[1] Malicious rootkits are used to hide inappropriate actions from the OS and anti-virus software. There have been a number of research efforts into attempting to classify different types of rootkits [39, 50].

Timing Attack This form of attack exploits sequences of system calls to find vulnerable states and use knowledge of the interval an IDS scans at to avoid detection.

Relocation Attack This attack consists of malicious code which relocates itself to avoid detection. Code can be relocated to memory which is not monitored or even potentially to remain purely in cache [55].

Rutkowska presents a taxonomy for defining stealth malware [62]. Although she considers malware to only include code that modifies the behavior of the OS or applications sensitive to security, her taxonomy includes a *Type 0 Malware* that encompasses this type of malicious code. Her taxonomy is:

[1] Rootkits can be used for both beneficial and detrimental purposes, this research is primarily concerned with Malicious rootkits

Type 0 Malware This type of malware includes malicious code such as a botnet. She does not include this in her definition of malware because, although malicious, it does not compromise the OS or running processes.

Type I Malware This malware is defined as code that hooks into the OS or process by modifying relatively static resources, such as executable file and code sections in memory. She proposes that this type of malware can be detected by verification software. To accomplish this, there must be some baseline, such as digitally signed executables. The current roadblock to detecting *Type I Malware* consistantly is the practice of legitimate software, such as antivirus programs, using this hooking technique also.

Type II Malware This malware hooks into dynamic resources, such as the data sections of processes. Since these resources are supposed to be modified, a static verification tool cannot reliably detect this type of malware.

Type III Malware This malware does not hook into either the static or dynamic regions of the OS or processes. She presents her Blue Pill proof-of-concept, which uses AMD's®hardware virtualization technology. Since this type of malware does not modify any part of the OS or its processes, even a dynamic verification tool would not be able to detect the malware's presence. Blue Pill is discussed in more depth in Section 4.2.5.

2.1.2 Taxonomy of Vulnerabilities. Bazaz and Arthur present research towards creating a taxonomy of computer vulnerabilities [6]. Figure 2.1 illustrates their

proposed taxonomy. Although this taxonomy presents a good start for classifying vulnerabilities, the three main classifications of main memory, input/output (I/O), and cryptographic resources are incomplete. This taxonomy could be made more complete by expanding main memory and I/O to include other volatile memory locations within a system such as the memory controller's address tables and any modifiable firmware like the Basic Input/Output System (BIOS).

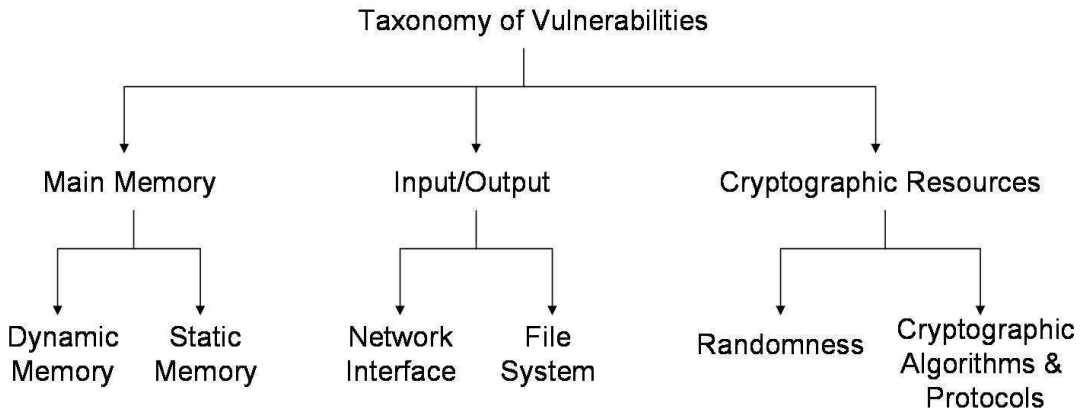

Figure 2.1: Bazaz and Arthur's Taxonomy of Vulnerablities

2.1.3 Important Specific Exploits. This section presents specific exploits to vulnerabilities that motivate aspects of this research to go beyond the standard areas of security discussion.

2.1.3.1 Defeating Hardware Based RAM Acquisition. Rutkowska discusses both software and hardware approaches to memory acquisition with the claim that the hardware-based approaches are superior to that of software-based solu-

tions [63]. After citing non-persistent malware as motivation for needing memory acquisition, she presents a number of known exploits of software memory acquisition by code running at the same privilege level as the acquisition software and notes that they require additional software on the target machine which she claims violates the forensic tool requirement not to write data to the machine which is targeted. Rutkowska extols the virtues of hardware-based solutions, setting her readers up for her defeat of this "superior" memory acquisition method.

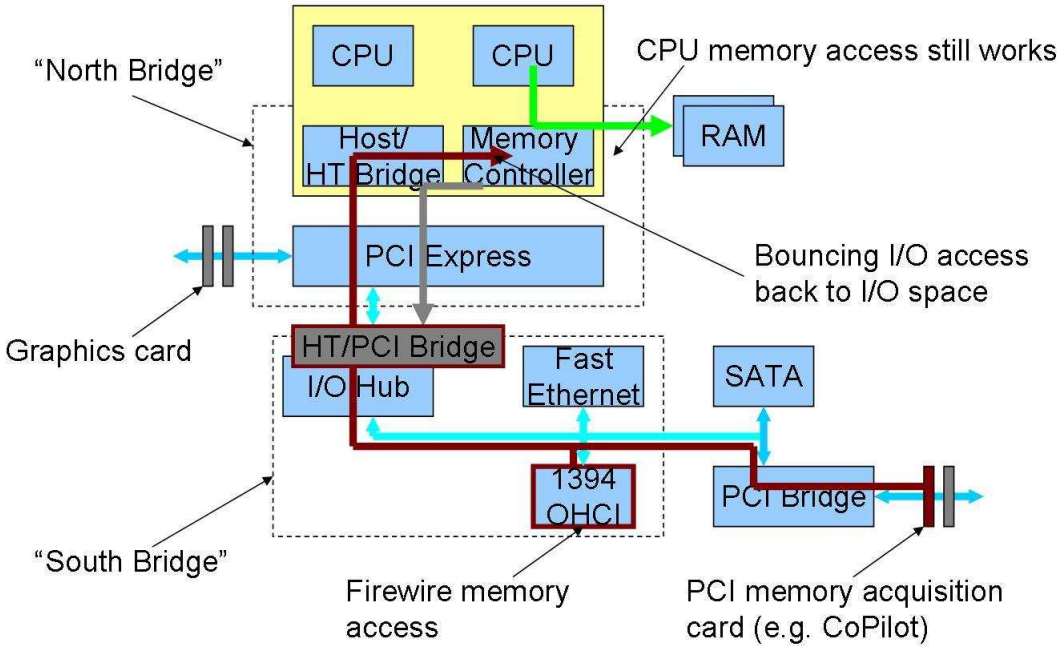

Figure 2.2: Rutkowska's Defeat of Hardware Based RAM Acquisition

Rutkowska delivers three levels of compromise to hardware based memory acquisition devices such as CoPilot and Tribble each building upon the same basic exploit with increasing levels of damage. As shown in Figure 2.2, this exploit involves configuring the memory controller on the north bridge to map arbitrary ranges of physical memory to I/O space. This remapping denies memory access to peripheral devices,

in I/O space, for the specified physical memory range while not effecting the memory access of the processor(s). The three levels of exploitation she presents are:

Denial of Service Attack Her attack can deny the monitor access to the specified memory range.

Covering Attack It can provide the monitor with repetitive masking data.

Full Replacing Attack This attack can even provide the monitor with specifically formatted data to deceive the monitor into believing it is monitoring a system which has not been compromised.

The exploits which Rutkowska presents definitively show that current hardware based memory acquisition devices, specifically those which plug in as a PCI device, are not reliable. The lesson to be taken from her work is not that hardware cannot do a better job of providing security features, rather that hardware is not a magic bullet; it does not automatically improve security. This work highlights that many current hardware solution are missing an important aspect of the capability and security of the monitoring system. This provides a substantial motivator to explore the trustworthiness of the information being received by a security monitor. This critical axis of security for a monitor, though acknowledged in numerous works [36, 79, 55, 11] is not well understood and certainly not clearly defined. Section 3.2.3 provides definitive categorization along this axis to help all future work in security related fields understand what is required to provide truly reliable security monitoring.

2.2 Taxonomies and Classifications of Intrusion Detection

Significant work has gone into classifying the different aspects of intrusion detection. This section discusses many of these taxonomies to provide a framework for classifying our security backplane as well explore missing classifiers from current work. It discusses classifications for how a system attempts detection, the goals of detection, the timeliness of response of detection, and the response itself. This section finishes the discussion of classifications and taxonomies by discussing how to classify the security of the IDS itself. A second axis of the IDS's security not well explored and previously undefined is noted, relating to the trustworthiness of the information the IDS is receiving.

2.2.1 Intrusion Detection Methods. Axelsson [5] describes a taxonomy of intrusion detection, which is extended by Williams [79] to include specification-based attacks first described by Ko [35]. This taxonomy is:

Anomaly-based Detection This type of detection monitors a system or process for abnormalities. It assumes anomalous activity is most likely *non-self*. There are two types of anomaly-based detection: self-learning and programmed, which are differentiated on how they establish a baseline of *self* to compare against for abnormal behavior. These generally have better ability to detect novel attacks.

Signature-based Detection This detection checks potential intrusions against a database of signatures. Signatures can be based on both the actual intrusive code or the traces left in a system from them. They operate without knowledge

of what constitutes normal behavior in a system. These signatures must be very specific and have much better coverage rates from *known* intrusions.

Specification-based Detection This type of detection provides capabilities to systems which have clearly defined security specifications. These systems can build models to detect deviations from these specifications. These specifications take two forms: default-deny, which specifies *legitimate* actions; and default-permit, which specifies *illegitimate* actions. Williams points out that although Axelsson categorizes much of this as anomaly-based detection, it shares elements of both anomaly-based detection and signature-based detection, and does not fit neatly into either category [79].

Each of these types of detection provide valuable abilities to a system. The inclusion of a dedicated security processor provides for increased ability to analyze the system's state for anomalous behavior and provides a protected location to store signatures. The increased precision of detection and response provided by our system allows for significant improvement in the use of specification-based detection, since the specification can be defined more precisely. Specifications can be securely updated and are protected from malicious tampering. This is important since any compromise to the detection methodology can invalidate its ability to function correctly. One example of this would be a specification on allowable jump targets for a specific section of a process' code. If malicious code can modify what targets are allowable, it can mask its intrusion by adding the malicious jump target to the allowable list.

2.2.2 Goals of Detection. Kuperman [36] puts forth different goals of computer security monitoring (CSM) in his dissertation. These goals are:

Detection of Attacks This involves detection of attempts to exploit a specific vulnerability.

Detection of Intrusion This involves detection of non-legitimate users attempts to exploit the system.

Detection of Misuse This involves detection of inappropriate use by authorized user

Computer Forensics This involves data gathering of previous activities to attempt to capture what caused departure from a safe state

Our security backplane enhances the ability of a security system to achieve each of these goals though the largest thrust of our research focuses on the detection of attacks. The addition of hardware monitors throughout the system provide access to information which is not normally gathered and can be leveraged against specific vulnerabilities. Though it is not the focus of our research, the security backplane at the networked level can provide vital resources for computer forensics.

2.2.3 Timeliness of Detection. Kuperman also uses the timeliness of detection to help classify the operation of different CSM systems. His notation recategorizes time into an ordered sequence of events. With this understanding he defines the set of all events that can occur in the system as E, the subset of all malicious events as $B \subseteq E$ and three events $a, b, c \in E$ and $b \in B$ Given the notation t_x to represent the

time of event x occurring and $x \rightarrow y$ representing a causal dependence of y upon x, we assume that they three events are related such that $a \rightarrow b \rightarrow c$ and therefore

$$t_a < t_b < t_c \tag{2.1}$$

must be true. Note that although $x \rightarrow y$ represents a causal dependence it does not necessarily mean that x is the direct cause of y.

The last piece of notation Kuperman defines for these purposes is the detection function $D(x)$ which determines the truth of the statement $x \in B$. This detection function is both complex and widely varied through different security systems and are almost exclusively imperfect. Kuperman defines the two problem cases for detection within the bounds of his notation

$$\text{False Positive: } x \notin B, D(x) = \textit{true} \tag{2.2}$$

and

$$\text{False Negative: } x \in B, D(x) = \textit{false} \tag{2.3}$$

With the notation defined Kuperman presents four main timeliness categorizations for CSM. These are:

Real-time Detection The detection of a bad event, b, occurs while the system is operating and before any event which is dependent upon b occurs. Therefore

the following order is required

$$t_b < t_{D(b)} < t_c \tag{2.4}$$

or alternatively,

$$t_{D(b)} \in (t_b, t_c) \tag{2.5}$$

Near Real-time Detection The detection of a bad event, b, occurs within some predefined time step δ, either before or after t_b.

$$|t_b - t_{D(b)}| \leq \delta \tag{2.6}$$

or,

$$t_{D(b)} \in [t_b - \delta, t_b + \delta] \tag{2.7}$$

Periodic Detection Also referred to as Batch Analysis, batches of events are analyzed by the security system at a periodic interval, p, which is normally on the order of minutes or hours. Therefore events must be ordered

$$t_{D(b)} \leq t_b + 2 \times p \tag{2.8}$$

giving the security system, in the worst case p time to analyze the batch. If $t_D(b)$ violates this constraint, the security system will not be able to finish its analysis of a batch before the next batch analysis needs to start.

Although near real-time and periodic detection have effectively the same measure (with δ being equivalent to $2 \times p$) the key difference between the two is that near real-time is *event* driven, where periodic driven by the security system and polled at the rate p.

Retrospective Detection The detection of a bad event does not occur within any set time-bounds and typically use archived event data. Many systems which operate within the first three time categories also have the ability to operate in this manner as well.

Kuperman comments that this timeliness categorization should be independent of the underlying hardware and the rate of event occurrence. Although this goal is desirable for a software-based solution, it relies on assumptions of trustworthiness and lack of vulnerabilities in this underlying hardware. With today's computer hardware this independence is unobtainable. One specific example of why hardware cannot be blindly trusted is Rutkowska's attack discussed in Section 2.1.3.1. Our research aims to achieve Real-time Detection and significantly reduce the δ value for Near Real-time Detection when Real-time Detection is not achievable.

2.2.4 Intrusion Detection System Responses. Stakhanova et al. present research towards defining a taxonomy of intrusion response systems [68]. Figure 2.3 presents their taxonomy discussed briefly here. From this taxonomy they highlight key categories which are desirable for an "ideal intrusion response system"

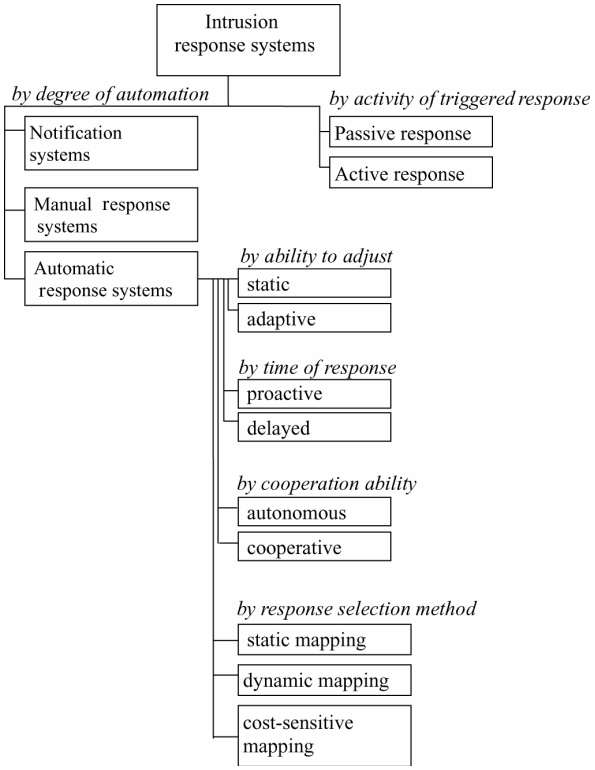

Figure 2.3: Taxonomy of Intrusion Response Systems copied directly from [68]

Automatic Due to the volume of intrusions and the speed with which a system can incur serious damage from being compromised, solely human based responses have an unacceptably high window of vulnerability. The more the response can be shifted towards automated, the better the system is able to respond.

Proactive A system's response needs to occur quickly to minimize the impact of compromise. Ideally a system would respond in real-time (2.4) allowing the system to completely negate the impact of the attack.

Adaptable The more a system is able to adapt to the the changes within a system and changes to the methods of attack, the more the system will be able

to remain autonomous from administrator interaction and continue responding automatically effectively.

Cost-sensitive This category of system is designed to take into account the idea that the intrusion system's response might be more costly than the actual symptoms would be. A system which takes into account this tradeoff can provide a more tailored response to potential attacks.

Our hardware-based security backplane, described in Chapter V, incorporates many of these concepts. Our design is shaped to fall within most of Stakhanova's desirable categories.

2.2.5 Classification of Monitor's Security. An often overlooked aspect of a computer security monitor is the security of the monitor itself. This security is a critical aspect of a security system, since compromising the monitors can effective render the security system useless. Mott presents a classification of the security of the monitors creating eight levels of monitoring system security [48].

Open This worst case scenario occurs when the monitored system has knowledge of the monitor and coordinating and sharing information with the monitor without any security mechanisms present to protect the monitor from compromise.

Soft Security This level of monitor security is equivalent to *open* with additional software techniques used to secure the monitor. Both of these levels tend to contain monitors on a uniprocessor host-based intrusion detection system.

Passive Security The monitor operates without the monitored system necessarily knowing it is there. To compromise such as system, information about how the monitor analyzes gathered state data must be known. Prime examples of this level of security are most network IDSs where only network traffic is monitored. Specific information passed over the network has the potential to disable the system, but their are no direct avenues of attack. IDSs of this nature are discussed in Section 4.3.3.

Self Security Similar to both *open* and *soft* security systems, the monitored system shares information with the monitor. The manner in which the monitor operates provides it with security, requiring the monitored system to be compromised before the monitor can be compromised. This security can be enhanced through software-based techniques. An example of this level of security is Williams' CuPIDS [79] which is discussed in Section 4.3.2.2.

Loose-hard Security The monitored system again has knowledge and coordinates with the monitor, sharing information, but dedicated hardware mechanisms protect key portions of the CSM from compromise. One example of this level of security are hardware-based return address stacks [38] discussed in Section 4.3.4.2.

Semi-hard Security The monitored system's knowledge of the monitor is extremely limited. To provide this level of security the monitor cannot execute on the same processor core as the monitored software and communications happens through mechanisms like unmaskable interrupts which are kept to a minimum.

Compromise can only occur via code controlling synchronization signals to the monitor, which would cause the monitor to operate in a diminished capacity.

Strict-hard Security This security level adds to the requirements of *semi-hard* security by requiring only hardware connections to the monitor and removing synchronization signals to the monitor. The monitor must be able to gather its own state information to remove dependence of the monitor on the monitored system. Two examples of this level of security are CoPilot [55] and Independent Auditors [47]. Two examples of this are discussed in Section 4.3.1.2 and Section 4.3.1.1 respectively.

Complete Security This level of security is the ideal secure case, used as a theoretical comparison point. In reality, such a monitoring system would have no contact with the outside world, making it self defeating because it is unusable.

Mott notes that within many of these levels of security, there is a tradeoff between the security of the monitor and the ease with which state information can be gathered from the monitored system [48]. One critical piece of information overlooked by these categories is the trustworthiness of the information that the monitor is receiving. Although technically the monitor itself is not corrupted, the effects can be equivalent. For example, a Supervisory Control And Data Acquisition (SCADA) System controlling critical infrastructure such as the electrical grid, could be manipulated to perform undesirable actions, without ever compromising the SCADA System. This can still be accomplished by an attacker who can only manipulate the information being received by the SCADA System. For instance, if an attacker can manipulate the

information feeding the SCADA System, telling it that there is a massive overdraw on the electrical grid, they can affect SCADA System responses such as causing a rolling blackout. This is accomplished without specifically corrupting the SCADA system to do so. The SCADA System would respond correctly to the environment it believes exists, not the actual environment. A simpler exploit corrupting the information being passed to monitors is a denial of service (DoS) attack. If the SCADA system does not receive readings from sensors monitoring critical sections of the system, it will be unable to respond to parameters out of acceptable ranges. This could quickly compound into catastrophic failure.

Although this issue is acknowledged in relation to ID in a number of works [79, 15, 55], little research has been found that delves into this aspect. Our research explores this aspect of the monitor's security. Rutkowska presents methods for corrupting the memory access of the PCI Bus without effecting the processor's access to memory [63] which is discussed in more detail in 2.1.3.1. This exploit highlights the importance of this aspect of classification for the security of the monitoring system. Section 3.2.3 presents an independent axis for categorizing the security of the monitor relating to the trustworthiness of the monitored data. This categorization looks at whether the monitoring device relies solely on the component it attempts to monitor or must trust intermediate components to pass it information. CoPilot, one of the examples Mott identifies as being *strict-hard* security, is defeated by this attack because of its security weakness on this new axis of categorization.

2.3 Trusted Computing

This chapter concludes by discussing the concept of trust in computing. This section looks at definitions of trust and a number of proposed requirements for achieving trust.

2.3.1 Trust Definitions. Significant work has gone into trying to define and model trust in computer systems. Numerous models abound each with their own take on how best to quantify trust: hTrust [10], VTrust [57], Secure Environments for Collaboration among Ubiquitous Roaming Entities (SECURE) [9], security and trust enhanced mobile agent (SATEMA) [18], an Architecture for Mobile Agents with Recursive Itinerary and Secure Migration (MARISM-A) [59], and I-TRUST [72]. VTrust is discussed in more detail, both due to the relative merit of the Trust Vector model and Stevens' work applying the use of trust vectors to the Cybercraft Initiative [70].

Ray and Chakraborty develop a Trust Vector model with three main components of trust: experience (W_e), knowledge (W_k), and recommendations (W_r) [57]. Using these vectors they define a range of trust from +1 (complete trust) to -1 (complete distrust) with 0 representing no trust. Since these three vectors are used to represent the entirety of trust in their model it holds that $W_e, W_k, W_r \in [0, 1]$ and $W_e + W_k + W_r = 1$. To generate the overall trust of a remote agent we take the experience evaluation ($_AE_B^c$), the knowledge evaluation ($_AK_B^c$), and the recommendation component ($_\psi R_B^c$) with each of their respective components. Given that $_AE_B^c, _AK_B^c, _\psi R_B^c \in [-1, +1]$, we know that $W_e \times_A E_B^c + W_k \times_A K_B^c + W_r \times_\psi R_B^c \in [-1, +1]$. Their work also discusses a

trust decay factor. As time passes, after the trust vector of a particular agent has been evaluated, the level of trust becomes less and less certain since the agent is vulnerable to compromise this entire time. One flaw in the equations for this degradation, is that they do not take into account any level of confidence in the security of the agent being trusted. It stands to reason that the better protected and secured an agent is, the higher level of confidence that we can have on continuing it's current level of trust.

2.3.2 Roots of Trust. The Trusted Computing Group (TCG) defines the concept of the *root of trust* as system components which must be trusted to guarantee detection of compromise [24]. They present three common *roots of trust* in trusted platform:

Root of Trust for Measurement (RTM) This is responsible for providing the basis for trusting integrity checks of the system and the continuous security of the system.

Root of Trust for Storage (RTS) This is responsible for providing the basis for trusting information stored by the RTM.

Root of Trust for Reporting (RTR) This is responsible for providing the basis for trusting the mechanism which allows reporting information stored by the RTS.

Our SHI(EL)DS architecture makes advancements towards developing each of these *roots of trust* in the system. Section 6.2.1.1 explores how this architecture

provides these advancements and relates it to the development of a Cybercraft deployment platform.

2.3.3 Requirements for Trusted Security System. Anderson proposes the inclusion of a *reference monitor* as an essential element of a secure system model, such as a *reference validation mechanism* [2]. Figure 2.4 displays the basic architecture concept of a *reference monitor*. Three essential elements to the proper design of these *reference monitors* are listed here.

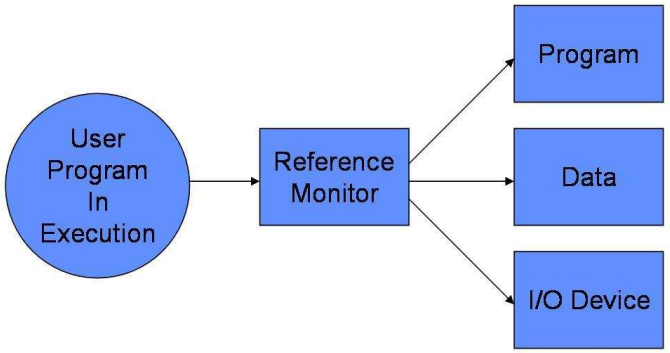

Figure 2.4: Reference Monitor

Must be tamper proof To be able to guarantee the integrity of the *reference monitor* it must be tamper proof.

Must *always* be invoked If the *reference monitor* is not always invoked, it cannot guarantee accurate monitoring of a system.

Small enough to receive *complete* analysis and testing The *reference monitor* must be simple enough to be able to prove that its design and operation are correct.

Our system makes headway towards each of these requirements, by designing a security system which can maintain these requirements. Although the production system will not be tamper proof or small enough to be completely tested, the physical separation of the security system will harden its resistance to tampering and the communication pathways between the two are simplified to the extent that complete analysis might be achievable. The dedicated security also provides the ability to always monitor its target, since the lack of resource conflict between the production and security systems allows the monitor to always be invoked.

III. Security Backplane Motivation

To motivate the need for a hardware security backplane we must first look at hardware in security. Is it necessary? What advantages can be gained through its use? What capabilities cannot be achieved without its use? What do we mean by hardware security? What design requirements are necessary for hardware to enhance current capabilities and achieve new ones? This chapter examines these questions to develop a detailed understanding of using dedicated hardware for security. These findings are used to examine a number of current and proposed security solutions in the following chapter. Chapter V leverages the understanding gained here to develop a hardware security backplane architecture, called SHI(EL)DS.

3.1 Research Hypothesis

Current computer architectures have been designed almost completely with performance as the primary goal. Creating a viable security system for today's computers requires modification of the basic hardware architecture of these systems. By creating a separate, parallel security backplane with limited connections to the production system this research demonstrates a system which provides necessary architecture modification to enhance security while allowing the functionality and performance of the production system to be minimally unaffected. This separation allows for significant flexibility in the implementation of security response to data gathered from the proposed hardware security monitors.

3.2 Why Hardware?

Most current security systems for computers are based largely on software systems. Numerous flaws and vulnerabilities have been exposed and even exploited in these different software solutions. Compromise of protected code via rootkits [39] represent one of the most prevalent exploits. Recent work has begun exploring different hardware-based approaches to security [11, 54, 23, 47, 48, 55, 81, 67, 82, 22, 28, 26, 8]. This research suggests that one cannot solely use software to protect software and only hardware, coupled with software, can do that job successfully. Though a number of advantages to hardware over software have been suggested, no research was found discussing what precisely makes hardware a significant improvement over software and just what capabilities hardware provides that software cannot. A number of key advantages achievable through the use of hardware are:

Reduced Avenues of Attack Separate monitoring hardware can strengthen the security of the monitor by reducing the extent of the coupling between the security and production systems.

Ability to Gather Trustworthy Information Correctly designed hardware guarantees that the monitor receives valid data from the production system. This cannot be accomplished with a standard software security solution, since it must rely on underlying hardware components.

Additional/Different Information Available Mott's research explores a number of pieces of information which can be gathered through hardware primitives and

leveraged to increase the overall security of the system [48]. These hardware primitives include information such as the program counter, instruction traces, and added visibility into memory.

Timeliness of Detection The ability to guarantee real-time detection, as defined in Section 2.2.3, requires the ability to guarantee that the monitor will execute with the ordering $t_b < t_{D(b)} < t_c$. Work such as Williams' CuPIDS [79], which at first glance appears to be a software only solution, does in fact take steps towards a hardware solution to achieve this ability. When CuPIDS is monitoring a process, it runs on dedicated hardware, one of the processors in the multiprocessor system which is a requirement for Williams' work.

Physical Response Another advantage of dedicated hardware is the ability to provide a physical response. Although this idea falls outside of this research and will not be further discussed here, the possibility for a physically destructive reaction in response to compromise could yield added security. Although software has the potential to destroy information, dedicated hardware can provide autonomous, timely response not achievable through software. Research into such dedicated hardware already suggests hardware-based destructive elements such as the patent filed by Vatsaas and Erickson [76]. This patent suggests responses to a stimulus such as providing an electrical charge and mixing chemicals to form a reaction. The proposed purpose of these responses is to "disable, damage, or and/or destroy" the component being protected.

The rest of this section develops justification for needing capabilities beyond what software can provide and explore each of these advantages in greater detail. It defines precisely what is required of hardware to overcome the vulnerabilities of software and provide significantly improved performance.

3.2.1 Vulnerabilities of Software Security Systems. There are a number of vulnerabilities inherent in software security. Two critical vulnerabilities are the inability to guarantee real-time monitoring in standard commercial operating systems, even on a multiprocessor system, and the inability to protect the integrity of the security system once the production system has been compromised. The first vulnerability is evidenced by the fact that scheduling of processes on both uniprocessor and multiprocessor systems does not make any guarantees on precise ordering or timing of when a specific process gets time on a processor. Work such as Williams' CuPIDS, discussed in Section 4.3.2.2, changes this standard paradigm to guarantee monitored processes run in lock step with the monitoring process [79] and overcome this first critical vulnerability of software security systems. Despite CuPIDS' ability to overcome this vulnerability, it cannot protect itself once the kernel has been compromised.

The specific point where software loses the ability to protect other software is when faced with exploitation of a vulnerability in privileged code. Once an attack can gain access through such a vulnerability, they have access to any piece of software in the system and can modify both data and executable code. This allows for changes in both user applications and the operating system itself, compromising the

security of the security system itself. This can be accomplished through modification to the security software itself or by modifying the operating system to interact with the security software in another manner, such as reducing its privilege level. Note that exploitation of vulnerabilities in privileged code provides two main avenues of attack into the system. The more obvious method of attacking the security software itself to degrade or interrupt its capabilities described above, but also corrupting the information that is being sent to the security software.

This second issue is the general method that rootkits use to remain undetected. They interpose themselves between processes by taking control when there is a library function or system call. By controlling what information is passed back to the process the rootkit can neutralize the security software without directly modifying it.

3.2.2 Advantages of Hardware. The vulnerabilities of software discussed above show clear need for a security solution which can overcome these vulnerabilities. Does hardware provide protection from these attacks? Not necessarily. Hardware can provide this protection, but only if appropriately designed into the system's architecture. Two key factors in designing hardware which can enhance these areas of security are where the security hardware connects to the system and how these connections are made. Where it connect controls the trustworthiness of information as well as influences the timeliness of detection. How the security hardware is connected impacts the amount of information available to the security system and defines the only avenues of direct attacks on the security system.

3.2.3 Trustworthiness of Information. Although the need for accurate data being received by the monitor is understood, there is no real framework for understanding what precisely is needed to accomplish this. Towards this end this research defines a new axis categorizing the trustworthiness of the information being received by the monitor. Although presented here towards the development of a hardware-based security backplane system, this axis of trustworthiness stands as its own contribution, which should be considered when attempting to provide an accurate and secure monitoring device of any sort. This categorization sets important bounds on what exactly affects the trustworthiness of the information.

Immediate Information (Figure 3.1) Immediate access to what is being monitored insures the monitor is receiving true data. This immediate categorization represents a specific form of first-hand information where the monitor is inline, directly between what is being monitored and its interaction with the system. While this level of trustworthiness is certainly the most definitive method for ensuring the monitor's security, it leads toward a design with individual monitors on every single hardware component, thus requiring a complete redesign of all aspects of a system.

First-hand Information (Figure 3.2) This level of trustworthiness represents a monitor that has direct access to the data being output from some device. Depending on the specific design of the architecture being monitored, this level of trustworthiness will likely be equivalent to *Immediate Information*. However,

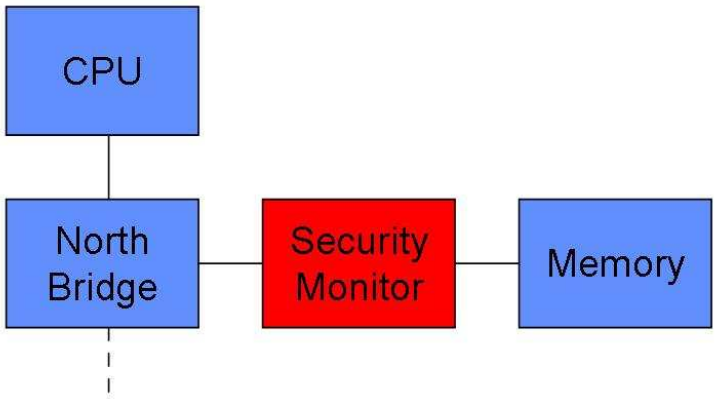

Figure 3.1: Immediate Information: Security Monitor placed inline between main memory and the memory controller.

a shared bus architecture could be vulnerable to DoS exploit. This would be accomplished in much the manner that someone would have trouble listening to another's conversation in a crowded room. For example, the PCI architecture provides a scenario where data for each system is broadcast to all connected systems. If a security monitoring system is connected to the hub to take advantage of this broadcast information it can be vulnerable to a DoS attack.

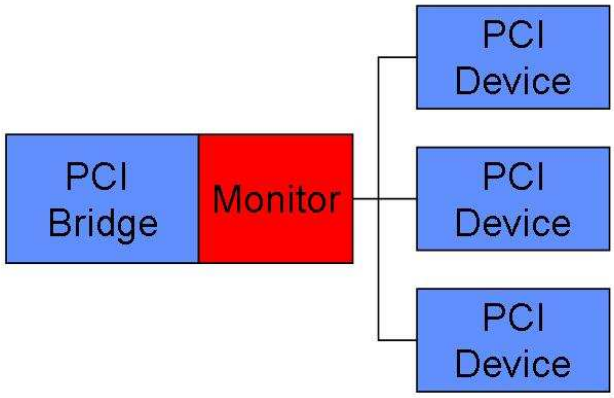

Figure 3.2: First-hand Information: Security Monitor placed on a shared bus, vulnerable to Denial of Service from excessive traffic.

With *First-hand Information*, or *Immediate Information*, a monitor is guaranteed to receive accurate data. The quantization from Section 2.3.1 shows that the only possible way for a monitor with worse than *First-hand Information* to guarantee it receives accurate data, is for *every* device which passes information to the monitor to hold a trust rating of +1. The potential vulnerability presented of a DoS does not stop accurate data from being presented, merely floods enough information to the monitor so that it is unable to interpret the correct data. The speed of the monitor as compared to the speed of the monitored information transmission plays a large role in this vulnerability and can even overcome it if the monitor can process information faster than it can be transmitted. If the monitor can process the incoming data at the same rate as the device it is monitoring, its monitoring still remains accurate, since a DoS attack on the monitor would effect the production system in the same manner. With the pre-knowledge of its operating speed compared to what it is monitoring, a monitor can recognize a DoS situation against itself as the same DoS against the production system component.

Second-hand Information (Figure 3.3) This level of trustworthiness encompasses any monitor that relies on some intermediary mechanism, such as hardware or software components, to pass it the data it is attempting to monitor. Each additional mechanism relied upon reduces the trustworthiness into third-hand information, fourth-hand information, and so forth. Each level having a continually lessening degree of trustworthiness. For simplicity this categorization

groups all levels of trustworthiness that cannot guarantee accurate monitoring into this category of second-hand information. Unless any and all mechanisms being relied upon to pass the monitor data can be guaranteed secure, this presents an avenue of attack for corrupting the monitor be feeding it false data. Figure 3.3 presents a simplification of Figure 2.2. It shows a PCI-based memory acquisition tool, such as CoPilot [55], that must trust the PCI bridge, the south bridge, and the north bridge; trust which Rutkowska's research demonstrates as unwarranted [63].

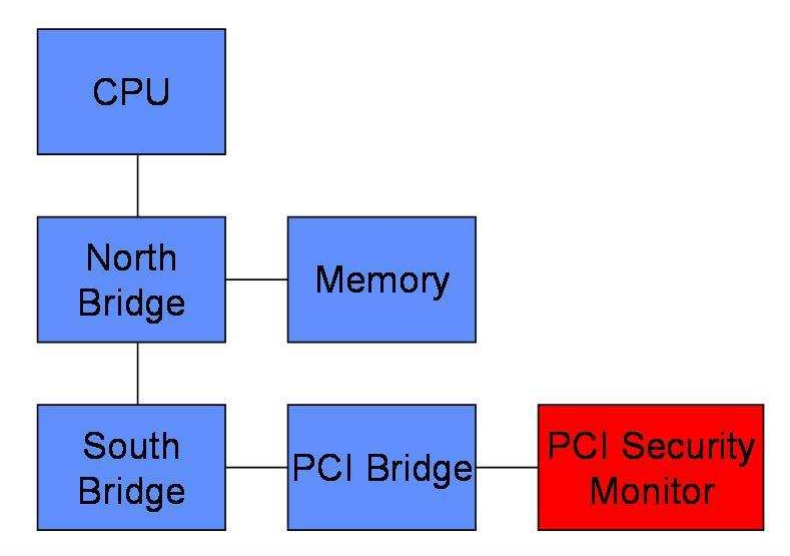

Figure 3.3: Second-hand Information:

Adapting the quantization of trust described in Section 2.3.1 to mechanisms within a computer system demonstrates the compounding problem of the trustworthiness of *Second-hand Information*. By assigning a trust value in the range $[-1, +1]$ to any mechanism that must pass information in a system it shows that the further removed from *First-hand Information* a monitor becomes the greater

the degradation of trust. Looking at the situation of Rutkowska's attack [63], by assigning arbitrary values of trust to the PCI bridge, the south bridge, and the north bridge of +0.9 each, that is all are 90% trustworthy, the aggregate trust degrades to 0.729, that is the monitor can trust that it receives accurate information with 72.9% confidence. This example assumes a fairly high level of trust for each component and still shows significant degradation. With trust of 80%, the aggregate trust reduces to only 51.2%. Note also that this example assigns trust values at a high granularity and does not deal with the aspect of the trust of each component on something such as a north bridge chipset.

It is this previously undefined axis of the monitor's security which is being exploited by Rutkowska's attack. This system incorporates this by attempting to ensure that the system is monitored predominantly by devices capable of receiving *First-hand Information*. Two important things to note about this axis of security are that 1) all software based security systems on a uniprocessor system are inherently unable to achieve a level of trustworthiness better than *Second-hand Information* since they must rely on data controlled by the operating system and 2) even software based solutions designed to operate within a multiprocessor system, such as CuPIDS [79], must still rely on the trustworthiness of main memory and therefore receive no better than *Second-hand Information*. In order to ensure accurate monitoring, the monitor needs to have access to at least *First-hand Information* of the data being produced, any intermediate devices provide the possibility of the data being manipulated before reaching the monitor. Therefore at very least monitoring or interaction points are

needed at each of the bridges in the system, i.e., any device that passes information from one part of the system to another.

3.2.4 Timeliness of Detection. Another aspect of monitor placement is the speed with which a monitor can detect an attack. One of the areas where the speed of a device far exceeds the speed of the buses which pass information to and from it is the processor(s). To accomplish real-time monitoring as defined in Section 2.2.3, monitors will need to be closer to the main processor than system bridges will allow. One clear example of why this is true is an attack which resides purely in cache. This section describes a theoretical cache attack in Section 5.1.1.3. Such an attack will be able to do its damage before detection, since detection is only possible with access to a present view of cache. Even accepting near real-time monitoring capabilities, Kuperman's δ value will be significantly smaller for a monitor which is located on-chip.

3.2.5 Hardware Primitives. The manner in which monitors connect to the system plays a significant role in enhancing both the security of the production system and the security of the CSM system. By limiting connections between the monitor and production systems and remaining within Mott's *Semi-hard* security level, the only avenues of directly attacking the security system are the hardware primitives that bridge the monitors and production system. As long as no primitives allow for modification of the monitoring system's code, a greatly reduced attack footprint for the security monitor is maintained. At the same time, these hardware primitives can offer direct access to information previously difficult to obtain and even provide

access to information not accessible through any software methods. Mott presents a number of hardware primitives that can be leveraged in [48]. The two main areas of interest for creating hardware security (and security in general) have been attempts to monitor processes running on the production system, mainly through various memory introspection techniques [11, 54, 23], and monitoring the incoming network traffic as it enters the system [67, 82, 22, 28, 26, 8].

3.2.6 What Do We Mean By Hardware Security? To this point we have left the definition of hardware security somewhat up in the air. All computer systems contain a mix of hardware and software and only a limited amount is accomplished with purely hardware. To create a security system purely in hardware would significantly hamper the flexibility and modifiability of such a system reducing the number of future attacks a system could potentially respond to. Solutions such as a field programmable gate array (FPGA) can be used to extend software flexibility into hardware, though it may require performance tradeoffs and is not pivotal to this aspect of the discussion. However, a pure hardware solution is not the goal when talking about hardware-based security. The key component of hardware-based security is the *communication between the production system and the security system.* Whether a specific monitor is pure hardware, a FPGA, or software running on some combination of hardware which remains separate from the production system hardware what qualifies a security component as hardware-based is that connection back to the production system. Note that an important result of this definition is that a hardware-based security solution

requires physically separated memory. This is not to say that pure hardware or at least FPGA solutions will not be required in some instances to provide fast enough response. Areas where high-speed detection is crucial will almost certainly benefit from pure hardware solutions. One predominant example of this is the network IDS field where research has shown benefits from hardware solutions [28, 8, 73, 22].

3.3 Specific Requirements for Achieving Benefits from Hardware

So far this research has discussed the different advantages of dedicated hardware for security solutions and discussed what is required to achieve these advantages. This section explicitly defines these requirements for dedicated hardware. By designing to these requirements, it is possible to design a comprehensive security solution that achieves the advantages of hardware previously explored. The SHI(EL)DS architecture discussed in Chapter V presents such a solution. These requirements are:

First-hand Information of all monitored information: This level of trusted information guarantees accurate monitoring of what is happening in the system. Without this level of trusted information security solutions are vulnerable to being denied access to the information or even fed false information. This vulnerability provides a route to compromise the effectiveness of the security system, without the need to compromise the security system itself.

Dedicated Monitors for parallel, concurrent monitoring: To protect against potential timing attacks monitors must be able to run concurrently with what they are monitoring to allow the possibility guaranteeing of Kuperman's *real-*

time detection. Any monitor which does not run concurrently with its target must ensure that it runs often enough to be impervious to timing attacks. In a software-based solution this becomes infeasible due to the performance penalty of continuous context switching. Dedicated hardware monitors remove the burden on production resources and keep performance degradation to a minimum.

Explicit Hardware Communication between the production and security systems: By limiting communication between the production and security systems to hardware pathways, avenues of attack upon the security system are reduced to these explicitly defined pathways. Without modifiable communication pathways, the ability to corrupt these pathways is reduced. These limited pathways provide a clear set of attack avenues which can be understood and protected.

Dedicated Storage of security code and data: Without dedicated, separate security storage software communication pathways remain present in the system. These communication pathways represent a significant avenue of attack to be exploited. Any software-based separation becomes vulnerable to a root-level compromise of the production system. Separate storage which cannot be directly modified by the production system provides a more reliable method of protecting the security code and data.

Dedicated Security Processor for controlling and coordinating the security mechanisms: Though not explicitly a requirement for gaining security capabilities, a dedicated security processor is included here for the coordination and communication abilities it can provide. This separate processor will allow for a secured

security control center when coupled with these other requirements. It will provide the ability to modularly add security mechanisms into a security backplane. An important aspect of this ease of modularity is the ability to combine both network IDSs and host-based IDSs into a combined, complete IDS which can leverage combined knowledge from each to provide more flexible and effective response.

3.4 Towards a Hardware Security Backplane

There are a number of key goals and factors which led to the creation of the hardware security backplane system concept. This section presents and discusses a number of these points that make the backplane concept the best solution with the current state of computer system architectures. Though other systems do provide some advantages, this system provides a more comprehensive and complete set of advantages to other systems, discussed in detail in Chapter IV.

Backwards/Forwards Compatibility By creating the bulk of the security system out of band, with only hardware primitive monitors as the connection, this architecture allows for little to no change to the software and firmware on a system. This means that a system equipped with this security backplane would be able to run the same code as a standard system without a security backplane. This also means that future development does not need to account for the backplane's existence, thus providing a high level of transparency to developers.

Modularity This backplane design provides an out-of-band system which connects to the production system only through specific monitored points. The backplane is designed to be capable of operating with only a subset of these monitors active. If a specific system only has access to monitors of the memory address allocation within the north bridge and the memory address requests on the PCI bus it can provide simple comparison analysis to ensure PCI devices receive the memory address they requested. If a specific system also has access to the network interface, it can provide the ability to shut down this system's network communication when it detects that the system is potentially compromised.

It also provides for a method of linking the backplanes from a number of systems together in a network that is almost completely separate from the global Internet like that of Figure 5.4[1]. This creates a system which can be leveraged in numerous ways. It can operate on a single or small number of systems, such as computers serving as the gateway into an internal network or providing external services (webmail, remote desktop, etc.). It can also operate as an entire network backplane to a server farm where individual backplanes can communicate potential threats throughout this network and control access to potentially compromised machines. This network level modularity and ways to leverage it for numerous security enhancements are discussed in Section 5.2.2.2.

[1]The only path of attack from an external network to the backplane network would be through a specific production system, through the hardware monitors of that system, through that specific security backplane out into the rest of the security backplane network.

Minimizing Impact By minimizing changes to the production system, redesign costs are reduced. This includes both the time and money required for redesign. The less modification required to the production system, the more independent development of the production and security systems can remain. The other aspect of impact is that of system performance. High-speed areas of system architecture will need careful design to minimize fanout and possible gate delays so as to keep these components operating within the correct clock speed. In some cases, the impact can be reduced to almost zero by creating a monitor which plugs into the system inline with a system bridge. In other cases, significant impact on the production system design will be unavoidable such as the inclusion of a monitor which would give *First-hand Information* of a system bus, such as the Intel®Front Side Bus (FSB) [30], to the security monitor. Determining where monitors are placed within a system relies on a cost-benefit analysis of the importance of information gained versus the extent of the impact to the system. In cases where the impact is insignificant, monitors which offer only a small benefit can still be realized. In those cases where the impact is greater, such as *First-hand Information* capable monitors of processor(s) and/or memory, the hardware primitives that can be monitored will need to be limited to only those which provide the optimal benefit.

Fits Design Requirements As discussed in Section 3.2, to create a hardware-based security system with a minimal attack footprint, the operation of the security system must be separated onto different hardware than the production system

maintaining only the necessary hardware primitive connections. This backplane design fulfills this need. The backplane architecture incorporates each of the requirements presented in Section 3.3. Since separate hardware is desired for the security system, this work attempts to only modify the production system with the inclusion of monitors. Section 5.1 explores critical monitoring points throughout the entirety of production systems and highlight specific hardware primitives to aid in this monitoring.

IV. Review of State of the Art Computer Security Solutions

Having presented requirements for hardware-based security solutions to overcome the vulnerabilities of software solutions, we now review the state of security solutions. We look at software, virtualization, and various solutions which include hardware to varying extents. We evaluate these solutions against the different advantages that the use of hardware provides. Although all of these advantages are of interest, the *Reduced Avenues of Attack*, provided by separation of the security system and the *Trustworthiness of Information* are two areas where most current solution fall short. These solutions often do not manage to achieve *First-hand Information*. When they do they still rely on the production system's memory, leaving this avenue of attack open.

4.1 Software Intrusion Detection Systems

Software security solutions abound both in the commercial market [43,52,66] and the academic research community [78,25]. All of these solutions are vulnerable to the weaknesses described in Section 3.2.1. For all their strengths as total system security or network security systems, they are unable to guarantee the *Trustworthiness of Information* and are vulnerable to any compromise of the production system. Despite these weaknesses, there are a wealth of useful approaches and techniques that have been explored in software. As explained in Chapter III, hardware in security does not necessarily replace the need for software, but provides improved security for the security system and new information to be used by the security system. This section

presents a select few software-based solutions that can be adapted to work with the SHI(EL)DS architecture. Numerous other software security systems exist which also share the potential to be adapted to the security architecture developed in Chapter V.

4.1.1 CDIS. Williams et al. develop a computer defense immune system (CDIS), which uses an artificial immune system (AIS) to address two main concerns of ID: the ever changing nature and the enormity of the network landscape [80]. This research is designed to augment traditional signature-based detection systems. They choose to implement an AIS due to the similarities between IDS structure and the biological immune system of humans. They use a process called *costimulation* to reduce the false positive rate by only discarding incoming packets if multiple antibodies detect it as *non-self*. The term *non-self* refers to any packet that is not considered acceptable network traffic.

4.1.2 jREMISA. Haag et al. present work on a multi-objective evolutionary algorithm (MOEA) inspired by an AIS that has the name jREMISA (java-based Retrovirus Evolutionary Multiobjective Immune System Algorithm) [25]. This work combines work by Edge et al. [16] and Coello and Corts [14]. Their work leverages the concepts of AISs and MOEAs to create an adaptive IDS which attempts to evolve *antibodies* which attempt to maximize the detection of *True Positives* while minimizing *False Positives*, as defined in Section 2.2.3. In Kuperman's notation a *True Positive* is defined as $x \in B, D(x) = true$. That is, some event, x, which is malicious, is detected to be malicious.

4.2 Virtual Machine-based Security and Separation Kernels

The virtual machine monitor (VMM) was first developed in the late 1960s as an approach to multitasking on mainframe systems, and with the advent of modern multitasking OSs fell out of use through the 1980s and 1990s [60]. With the renewed and increasing popularity of the virtual machine (VM) in today's market, for both platform flexibility and security, research has begun to focus on the strengths, vulnerabilities, and feasibility of using VMs for security purposes. This section explores elements of this research.

4.2.1 Virtual Machine Monitor Description. There are a number of security enhancements inherent in the design of VMMs [58]. A VM running a mainstream OS on top of a VMM has an extra layer of abstraction and protection. If the OS on this VM is compromised by an OS specific attack, only that particular VM is corrupted. An attack meant to compromise the entire system must be designed to bypass the initial OS and exploit a vulnerability in the VMM and attack the underlying OS if one exists. Robin and Irvine describe two basic classes of VMM: *Type I VMMs* and *Type II VMMs*. *Type I VMMs* operate as the direct link to the hardware, functioning as a stripped down OS and allocating resources to the different VMs running on it. A *Type II VMM* runs on top of a standard OS as an application.

4.2.2 Rootkit Defense. Medley presents research into using hardware assisted virtualization (HAV) to protect against rootkits [45]. The core of this research revolves around creating a small dedicated *Type I VMM* which is simple enough to be

designed securely. This VMM secures system resources by creating a new operating mode. Using HAV, Medley proposes a system which leverages the new advantages provided to remove most of the OS operation from root level without significantly impacting system performance.

4.2.3 Terra. Garfinkel et al. present a Trusted Virtual Machine Monitor (TVMM) architecture which aims to allow a variety of applications, *with varying levels of security needs*, to operate concurrently on a general purpose computer platform [19]. This TVMM, called Terra, operates as a *Type I VMM*. One of the key concepts which separates Terra from a standard VMM is the inclusion of both *Open-box VMs* and *Closed-box VMs* in the architecture design. An *Open-box VM* is the standard concept of a VM with the VMM and underlying OS aware of the activities within the VM. A *Closed-box VM* attempts to operate as a complete black box with regards to the rest of the system, denying access to the underlying OS for inspection or modification. This is done through encrypting all memory and storage for the *Closed-box VM*. Terra also includes a *management VM* which provides the ability to fine tune how different VMs are granted access to hardware. Garfinkel et al. identify a number of key strengths to their architecture:

Isolation Applications on different VMs remain well isolated.

Extensibility Applications with significantly different security requirements can be implemented concurrently.

Efficiency Virtualization has shown to have negligible impact on performance [27].

Compatibility VMMs can run mainstream OSs such as Windows and Linux.

Security The simplicity of VMMs compared to an OS allows for the ability to have higher confidence the security of the software.

They also identify three features which they claim are unique to their Terra architecture. These features are described here:

Root Secure The inclusion of *Closed-Box VMs*, as described above, provide the ability to run code that even the root administrator does not have access to.

Attestation Terra provides the ability to cryptographically authenticate a VM as the source of information to other network systems. This authentication, called attestation, provides information about every aspect of the source of information from the hardware all the way up the software stack of the system. This ability allows systems receiving information, with the ability to decrypt this technology, from a Terra equipped system to make their own, informed decision on the trustworthiness of the data received.

Trusted Path In order to build secure applications you must have a trusted path [42] which Garfinkel et al. claim their TVMM provides.

4.2.4 Separation Kernels. Rushby presents research into separation kernels which are a variant of VMMs [61]. The key differences of the separation kernel are the ability of the kernel to provide varying hardware profiles to the different *regimes* (Rushby's term for the VMs) and the ability to emulate specific, limited hardware

communication pathways. From this description, the Terra TVMM discussed above is very similar to an advancement of the basic separation kernel concept.

4.2.5 Virtual Machine Monitor Vulnerabilities. There are a number of key vulnerabilities in the design of VMMs. First and foremost, the VMM uses the same hardware, so a compromise of any vulnerability in the VMM design can compromise the entire system and all processes running on it. Second, new architectures allow VMs direct access to hardware opening up avenues of attack which can bypass the VMM, even if the VMM remains secure. Lastly, although not a vulnerability of VMMs precisely, the possibility of virtual-machine based rootkits (VMBR) has been demonstrated by King et al. with their SubVirt concept [34]. Rutkowska also presents a potentially malicious VMM using AMD's®virtualization technology [64].

While many VMMs claim to improve their self-security (over that of a standard OS) through simplifying the core kernel of code, none of them claim to achieve guaranteed secure operations. By shrinking the size of the VMM as compared to a standard OS they are reducing the vulnerabilities, not eliminating them or even clearly constraining what precisely those vulnerabilities will be. In each of the VMM solutions developed above, a compromise of the core of the production system still leaves the security system vulnerable. Terra provides a high level of security between different VMs through emulation of hardware communication pathways between them. This security becomes compromised if the TVVM or the controlling VM become compro-

mised. The system developed in Chapter V moves this protection mechanism into actual hardware, further strengthening its security.

Hardware assisted virtualization (HAV) provides a number of performance advantages for running a VMM on a system. Both Intel® [20] and AMD®have [71] have released architecture modifications to include HAV in order to support the growing trend in virtualization. Although HAV provides performance advantages, it also opens up additional avenues of attack. With greater process control of hardware, malicious code which corrupts a process can achieve its intended affect more directly.

VMMs can also be used in a malicious manner. King et al. present their SubVirt system which attempts to install a VMM underneath a targeted OS, encapsulating this OS as a VM running on their VMM [34]. This virtual machine-based rootkit concept, allows for their attack to very effectively hide from the production OS, while having a high level of visibility and control over the now compromised system. Any software security system operating on the targeted OS will have little to no ability to detect this VMBR or defend against it. Properly designed hardware-based solutions can detect against an attack of this nature, since it's access to operations will not have to pass through this VBMR to monitor the system, like a software-based solution would have to.

Rutkowska demonstrates a hypervisor (VMM) that uses AMD's®Secure Virtual Machine (SVM) to co-opt control of a system by shifting the host OS into a VM running on her thin hypervisor [64]. This hypervisor, called Blue Pill, is only respon-

sible for limited control of the OS. This allows Blue Pill to have very little impact on performance and reduces the visibility of this hypervisor to detection techniques possible detection techniques. Since a virtualization based rootkit does not modify the BIOS, boot sector, or system files of the original host OS, Rutkowska claims that Blue Pill, and other *Type III Malware*, are virtually undetectable. She admits that it is possible for a system running on top of a hypervisor or VMM to detect that this is true. She points out that detecting that there is a hypervisor underneath the OS does not mean that one can detect whether or not it is a malicious hypervisor. She also claims that even a complete kernel integrity scanner that could verify both the static and dynamic regions of the kernel and able to detect *Type I Malware* and *Type II Malware* could not detect her Blue Pill. Even with this impressive theoretical verification tool, *Type III Malware* remains undetectable.

4.3 Hardware-based Intrusion Detection Systems

4.3.1 PCI Based Devices.

4.3.1.1 Independent Auditors. Molina and Arbaugh present independent auditors that check the integrity of the filesystem to determine if an intrusion has occurred [47]. A PCI card based coprocessor logs all changes to the filesystem and performs auditing calculations in a personal computer architecture so as to ensure filesystem integrity. A policy file provides the basis for defining what files and parameters are to be audited providing useful computer forensics capabilities. There

work defines eight properties which must be accomplished by a device for it to be considered an independent auditor

Unrestricted Access The auditor must have unrestricted access to devices to be audited and used by the auditor.

Secure Transactions The auditor must must be able to reliably and securely retrieve data from the audited system.

Inaccessibility The audited system cannot have access to the auditor's internal components.

Continuity The auditor must begin running at system startup when the system is still in a trusted state and remain in operation as long as the system is operation.

Transparency The auditors access to the system's devices should be as transparent as possible.

Verifiable Software The auditor's software must be verifiably trustworthy.

Non-volatile Memory The auditor must be capable of maintaining information through power failure or reboots.

Physically Secure The auditor needs to be physically secure.

This list of properties provides a good list of important elements for any hardware-based security system. These properties, either in the form listed above or a similar focus, are incorporated into the hardware security backplane system. These independent auditors provide an interesting set of capabilities; unfortunately they fail to

meet their second property, reliably and securely retrieving data from the audited system, in their system design which Rutkowska's attack demonstrates [63]. This failure is in large part due to a lack of understanding in the trustworthiness of their information source. By defining clearly this aspect of a security monitor's security, as Section 3.2.3 does, this research hopes to rectify this oversight. These independent auditors do manage to enhance their own security through using limited hardware pathways to connect to the production system.

4.3.1.2 CoPilot. CoPilot [55], developed by Petroni et al., is a coprocessor-based IDS. This IDS monitors the integrity of the host processor's physical memory space and looks for changes such as the installation of known rootkits which compromise the host processor's security. They assert that a coprocessor must meet six criteria to monitor a kernel at runtime effectively:

1. Unrestricted memory access
2. Transparent to what is being monitored
3. Operate independently of what is being monitored
4. Capability to process large number of operations
5. Sufficient memory resources to contain image of clean host
6. Secure reporting of security system state

CoPilot attempts to meet these requirements as a peripheral component interface(PCI) device. As a PCI card, its memory access is coordinated through the CPU

and peripherals on the system buses, allowing CoPilot to monitor memory without explicit communication with the production processor. This places CoPilot within Mott's tight-hard security category for security of the monitor. Despite this, Section 2.2.5 shows that in can be rendered ineffectual because of its reliance on *Second-hand Information*. While the integrity of CoPilot is not compromised by Rutkowska's attack, its ability to monitor memory is neutralized, producing a net effect similar to an actual compromise of the CoPilot system.

4.3.1.3 Tribble. Carrier and Grand present another hardware-based memory acquisition tool which also resides on the PCI bus [11]. This device provides very similar capabilities to CoPilot by capturing volatile memory. An important requirement of their system is that it must be installed when the system is in a known good state so that it can establish a baseline for the system.

Since Tribble is a PCI-based device, it is yet another example of a hardware security device which is vulnerable to Rutkowska's attack, described in Section 2.1.3.1, due to its inability to capture better than *Second-hand Information*. Tribble provides itself with a high level of security by limiting communication pathways from the production system to hardware-based monitors.

4.3.2 Complete Systems with Dedicated Hardware.

4.3.2.1 APHID. Hart presents an Anomaly Processor in Hardware for Intrusion Detection (APHID) which uses co-processing ID to offload the security

processing burden [26]. This research uses tightly coupled monitors with anomaly-based detection to defend against attacks such as distributed denial of service (DDoS) and buffer overflow attacks.

4.3.2.2 CuPIDS. Williams implements a CoProcessor Intrusion Detection System (CuPIDS) which leverages a processor in a multiprocessor system to act in lock step with the normally functioning processor as the coprocessor [79]. Using the uniform memory access (UMA) multiprocessor model, this system accomplishes ID and security policy compliance monitoring (SPCM). Williams' design allows for the monitoring processor to access virtual memory, since it is tightly coupled to the production processor, located at the same logical level. This access provides the monitor with information of both kernel-space and user-space, where the PCI-based solutions discussed previously can only access kernel-space.

This architecture requires an OS which is not compromised, since CuPIDS runs within the framework of a single OS. The main functionality of CuPIDS are the process pairs known as the CuPIDS Production Process (CPP) and the CuPIDS Shadow Process (CSP). In order to minimize the performance impact, the CSP monitors the CPP only at critical points which are inserted into the CPP based on events that can be used to detect intrusion. The CSP is initiated first, followed by the CPP with "hooks" from the CSP into the CPP's virtual memory.

CSPs can be designed to utilize both *anomaly-based detection* and *specification-based detection*. Detection happens quickly compared to other software-based security

systems and has the ability to notify of a CPP compromise or block the CPP from further execution. Due to it's fast detection time and access to the CPP's virtual memory there is the potential for a block of memory to be copied when dangerous system calls are made, so that damage done by a potential attack can be repaired.

The two biggest shortcomings of the CuPIDS architecture Williams' identified are performance degradation and vulnerability when the OS is compromise. Williams' estimates a 15% performance hit from that of a system not being monitored. This performance degradation is measured through a comparison of clock time, user time, system time, and throughput. Due to CuPIDS running on a single OS, compromise of the OS leaves the entire system vulnerable to compromise.

Despite being viewed as a purely software solution, the manner in which CuPIDS is developed takes the first steps toward dedicated security hardware, by co-opting a processor for security use. When a CSP is running, that processor is dedicated to security purposes. Although it begins down the path toward hardware security, it shares the same OS and the same memory as the production system. This leaves it vulnerable to compromise if the production system is compromised.

Although as a software solution, CuPIDS cannot achieve anything better than *Second-hand Information*, it relies on significantly fewer mechanisms than PCI-based monitors. The tradeoffs between the two in terms of their own security is interesting, since the PCI-based monitors receive *Second-hand Information* from a number of hardware mechanisms, only one of which has a demonstrated compromise against

it. When the CuPIDS system has a CSP set up and running it has hooks into the CPP's memory and must rely on the same hardware that the CPP is relying on for its information. This monitoring however, only maintains an advantage over PCI-based memory monitors with the assumption of a OS which has not been compromised, this remains the most significant weakness of Williams' work. Leveraging the other design elements of CuPIDS while overcoming the limitation of its reliance on the production OS will lead towards a formidable security system.

4.3.2.3 Security Enhanced Chip Multiprocessor (SECM). Shi et al. present a similar IDS to CuPIDS with the most important difference being that SECM includes a separate operating system for the monitor [65]. SECM also shares the Level 2 L2 Cache which allows the security system to monitor every production processer request for data from memory at the cache level.

With a shared L2 Cache, SECM achieves *Immediate Information* of processes executing on the production processor. Although SECM's use of hardware enables it to guarantee that it receives accurate information from production processes, the security system code is still stored on the same hardware as the production system providing a large avenue of attack. SECM attempts to reduce this vulnerability by keeping the security kernel as compact as possible and reducing the privileges of the production system. Since SECM has not been implemented in hardware, or compared against a system that is not being monitored, the performance penalty of using this security solution has not been fully explored.

4.3.3 Network Hardware Intrusion Detection Systems.

4.3.3.1 Efficient Packet Classification Using FPGAs. Song and Lockwood present research into combining Ternary Content Addressable Memory (TCAM) with the Bit Vector (BV) algorithm to provide multiple matches at gigabit per second (Gbps) speeds from the classifier of their network intrusion detection system (NIDS) [67]. Their system, called BV-TCAM, uses separate, dedicated TCAM which allows for the efficient execution of their matching algorithm. This system provides continuous monitoring of network packets, but still requires access to information stored in main memory. If their system's access to main memory is degraded or delayed, the system becomes significantly less effective. Though not explicitly stated in their work, their hardware seems designed to receive *First-hand Information*, allowing it to guarantee it is monitoring the actual network packets entering the machine.

4.3.3.2 FPGA-based Content Addressable Memories. Bu and Chandy present a NIDS which uses dedicated content addressable memory (CAM) to achieve processing of incoming packets at 2 Gbps [8]. Their system uses *Signature-based Detection* by loading signatures into a CAM array and passing incoming packets through one bit at a time. They use a keyword match array that keeps track of bit matches and signals when full word matches occur. This process evaluates incoming packets within a linear multiple of input stream size. This multiple is a constant dependent on the number of keywords. Their work presents a clear example of dedicated hardware pro-

viding significant improvements in the detection time of malicious packets, achieving detection speeds more than capable of keeping up with today's gigabit networks.

4.3.3.3 Reconfigurable Hardware-based String Matching. Hutchings et al. present research into using FPGAs to perform string matching for network ID [28]. Their research leverages compiling regular expressions into nondeterministic finite automata (NFA) and implement directly onto FPGAs, instead of first converting the NFA into a deterministic finite automata (DFA).

Their research presents impressive results compared to software-based network ID, especially as the size of the regular expressions for string matching grow in size. They manage to maintain relatively moderate CPU utilization and a fast response time regardless of packet size, compared to software solutions which require increasing levels of CPU utilization and continually degraded response time. The benefit that hardware grants this security solution is predominantly a matter of the timeliness of detection and the speed of response.

4.3.4 Security Mechanisms Assisted By Hardware.

4.3.4.1 Real-time, Parallel ID via Hardware-based Architecture. Mott et al. present research into a number of hardware primitives designed to aid in real-time, parallel ID [49]. Their research presents extensions of the CuPIDS architecture focusing on improving real-time monitoring of the production processor unit (PPU) through parallel ID using a shadow monitoring unit (SMU). The SMU snoops the FSB

and directly monitors PPU state information. It also provides for direct control signals back from the SMU to the PPU. Their early prototyping work is developed on FPGAs with the envisioned solution designed to use either FPGAs or non-reconfigurable hardware with dedicated software. Mott details a number of hardware primitives in his thesis to be used for gathering context-rich state information [48].

Multi-context Hardware Monitors Provides the monitor the ability to distinguish between processes executing on the PPU.

Execution Policy Enforcement Module Designed to prevent malicious code from executing.

Peripheral Access Control Enforces access to peripherals, keeping processes from accessing unintended peripherals.

Asymmetrically Partitioned Main Memory Grants the SMU access to the PPU's memory while preventing the PPU from accessing the SMU's memory.

MMU Co-opting Provides visibility into a process's virtual memory space.

Monitoring Using Multiple MMUs Extends MMU Co-opting to be able to monitor the virtual memory of processes not executing.

Though Mott's work does not explicitly state the need for *First-hand Information*, many of these monitors are designed to achieve it. Asymmetrically Partition Main Memory starts down the path of dedicated storage for the security system, though since it appears to be accomplished through permissions, not physical separation, there is still a potential avenue of attack on the security system exposed. This

work has limited dedicated processing power and uses explicit hardware communication between the PPU and the SMU. Mott's work meets most of the requirements outlined in Section 3.3 for hardware-based security to provide superior performance over purely software-based solutions.

4.3.4.2 Hardware-based Stack Protection. Exploiting weaknesses such as a buffer overflow, one very common form of attack involves rewriting information on the stack, such as a return address [53]. By overwriting a return address on the stack with an alternate address, an attack can execute malicious, injected code. Lee et al. propose a secure return address stack (SRAS) which is designed to defend against this form of attack [38]. Another hardware solution, SmashGuard, is described by Özdoganoglu et al. [54]. Both of these stack protection techniques utilize separate, dedicated hardware storage which allows for increased security of the monitoring system, since the production system cannot write to the SRAS. However, these systems include the ability to store information in main memory if their dedicated memory is unable to handle the depth of the return address stack. This provides a potential avenue of attack to systems using these security methods. By ensuring the stack grows large enough to force a write to memory, malicious code can exploit this vulnerability. These solutions gather *Immediate Information*, which guarantees that the monitor received accurate data from what is placed onto the stack. Some form of hardware-based stack protection should be included in any complete security solution. One of these or a tailored solution provides an excellent modular addition to the architecture

developed in Chapter V. To increase the strength of their security, their overflow storage should be located in dedicated security system memory, not the production system's main memory.

4.3.4.3 Microinstruction-based Monitoring. Ragel et al. present research into modifying microinstructions that implement high risk instructions [56]. This research leverages the microinstructions of the instruction set architecture (ISA). These microinstructions are not accessible to the software programmer. Since multiple microinstructions are used to achieve a single instruction from the ISA, modification of the microinstructions is possible while maintaining ISA compatibility. They present a number of examples such as buffer overflow attacks, fault injection attacks, and out of bounds memory address access.

Buffer overflow attacks are prevented through a hardware-based mechanism similar to that described in Section 4.3.4.2. Instruction path fault injection attacks are monitored by comparing the instruction memory to what is actually fetched. Data path fault injection attacks are prevented via a first in first out (FIFO) buffer which stores the write-back address from the instruction decode state and compared to the actual write-back location. Out of bounds memory address access prevention through comparison against a particular memory range.

Ragel et al. report an average performance penalty of 1.93% on applications tested and no greater than 15% overhead to area on-chip. They measure this performance penalty as the percent decrease of the clock speed. Their research can po-

tentially provide *Immediate Information* for specific aspects of processes on the CPU which they monitor. It involves separation of monitoring hardware and uses the modification of microinstructions to make the modification transparent to the ISA. Their system also implements a small amount of dedicated storage for the security system. While the system makes a number of improvements through it's dedicated hardware, it requires significant change to the architecture on-chip and does not interface with the rest of the system. Also, the modified microinstruction approach detects and prevents processes from behaving other than they way they were designed. If the OS becomes compromised, processes can be modified so that their new intended purpose has malicious consequences without triggering a response from this security system.

4.3.4.4 Control Flow Monitoring. Zhang et al. present research into control flow monitoring at the instruction level through modified hardware [83]. Their research uses two main methods for monitoring proper branching. The first examines the text of a process to understand all possible branch targets. The second involves examining a process executing in a trusted state. Zhang et al. expand their work to include monitoring of multiple branches for anomalous behavior detection among other improvements [84].

The main drawback to these security efforts is the reliance on training the system to each process that is to be monitored.

Arora et al. also present research into this field [4, 3]. Their work focuses on monitoring the program counter PC, the instruction register IR of the instruction

which is finishing, and the status information from the control unit of the pipeline. They provide two control signals with their work: a *stall* and an *invalid* signal. They aim their research at providing protection to three key properties:

1. Inter-procedural control flow
2. Intra-procedural control flow
3. Integrity of the executed instruction stream

Each of these types of monitoring receives *Immediate Information* of the PC, IR, and control unit status through the use of dedicated monitors. The system also incorporates both dedicated storage and utilizes hardware communication pathways to gather data.

4.4 Non-detection Oriented Computer Security

All of the discussion so far has focused primarily on intrusion detection, i.e., response to an attempted attack upon the system. Another field of computer security focuses explicitly on providing trusted modes of execution for software processes. The most notable in this field is the Intel® Trusted Execution Technology, formerly known as LaGrande [32,31,33]. Trusted Execution Technology offers a number of capabilities:

- Protected execution and memory spaces to process sensitive data
- Sealed storage to protect data such as encryption keys
- Protected Input and Graphics through encrypted communication
- Attestation to provide assurances that a process is correctly invoked

- Measured launch allows a higher level of assurance for platform configuration verification

They make use of a Trusted Platform Module (TPM) that provides dedicated storage for encryption keys and establishes the *root of trust* for the system for attestation purposes. While the Trusted Execution Technology provides many benefits to the security of a system by providing dedicated storage, the technology is not designed to handle ID tasks. Trusted Execution Technology only protects code developed to utilize it, which leaves systems open to vulnerabilities from software that has not been re-engineered to do so. Since protected code must be developed and employ the new capabilities correctly, systems are also vulnerable to inexperienced programmers not fully understanding what is required to invoke the Trusted Execution Technology and provide their code with a trusted environment. Our work attacks security from the detection and response angle, attempting to provide security to all processes running on a system, regardless of how they were developed. This helps to remove the burden of security from the average programmer.

V. SHI(EL)DS Design

This chapter develops the Secure Hardware-based Intrusion (Elimination, Limitation, and) Detection System (SHI(EL)DS) architecture, exploring locations where security hardware can strengthen the security of the system. This discussion remains high-level, focusing on general architecture. It shows advantages which can be gained through this architecture and ties the decisions back into the discussion of the benefits of including dedicated hardware in security solutions from Chapter III. Although numerous solutions were discussed in Chapter IV with varying levels of hardware involvement, this architecture presents a complete architecture, combining strengths from numerous different security solutions and attempting to overcome weaknesses present in each. Although the SHI(EL)DS architecture does not represent an impenetrable security system, it brings significant progress to a number of key vulnerabilities that plague current software solutions.

Dissecting our SHI(EL)DS acronym for understanding what at first seems to be nothing more than semantic trickery to create a good acronym yields key insight into the goals and capabilities of our proposed system. The secure hardware-based intrusion detection system all holds clear meaning related to our research efforts: we are attempting to utilize hardware to create a more capable and secure IDS. Though the term intrusion prevention system (IPS) is commonly used to refer to systems which do more than simply detect intrusion, the term prevention tends to indicate that security systems will proactively defeat all threats. We use the term *elimination* to include both proactive and reactive defeat of threats. An important capability

that our security backplane attempts to provide is the ability to deny a corrupted peripheral access to the system or even entire corrupted system from access to the rest of the security backplane equipped network. This *limitation* of an intrusion which cannot be completely eliminated, potentially allows a system to remain operable while containing the intrusion.

5.1 Critical Monitoring and Interaction Points

This section explores critical monitoring and interaction points in Intel®and AMD®systems since they are two of the most dominant hardware architectures deployed today. Despite the specifics of other architectures being different, the more general concepts can still be extrapolated from this research. Although much of the overall architecture design between Intel®and AMD®are very similar, there are a number of key differences pertaining to AMD's®development of the HyperTransport Bus®. This section begins by discussing the critical locations in the Intel®Architecture and then present the differences required by AMD's HyperTransport Bus®. In the long term, the AMD®focused information will be more representative of how this system would be deployed, since Intel is moving away from their current shared bus architecture towards an architecture similar to that of AMD's HyperTransport® [29].

Before identifying critical points, criteria for choosing what effective monitoring and interaction points should accomplish must be defined. Ideally, all information within a system would be monitored in real-time and guarantee that what each device is doing is what the monitor sees. Although this can be accomplished by modifying

every single component within the system architecture, it would be unrealistic in terms of both the cost and the obtrusiveness into the system. A number of key factors important to identifying critical points are presented here.

1. Value and accuracy of monitored information
2. Timeliness of detection
3. Obtrusiveness into operation
4. Impact on performance
5. Cost of integration

Each of these key factors maps back to the advantages gained through hardware and the justification for developing a security backplane architecture. New hardware primitives must meet the criteria for presenting trustworthy data, by providing at worst *First-hand Information* about the system and that this information can be leveraged to increase the capabilities of the security system. As these goals are pursued, care must be taken to examine the tradeoff between what a monitor provides and the negative impact it has on the system. Any interaction point which modifies the general operation of the production system or presents a serious performance degradation must present substantial gains to justify inclusion. Large changes to the operation of current devices in the system can quickly make this too device specific and cost prohibitive. If this system causes a significant performance degradation there will be great resistance to adoption of this system since very few people are willing

to trade lower performance for increased security unless there is a large disparity in favor of the increased security.

5.1.1 Basic PCI-Express Era Intel System Architecture. Figure 5.1 presents a general layout of a desktop/server computer architecture. Although the specific details of what devices are attached in a system vary greatly, the general architecture remains fairly consistent.

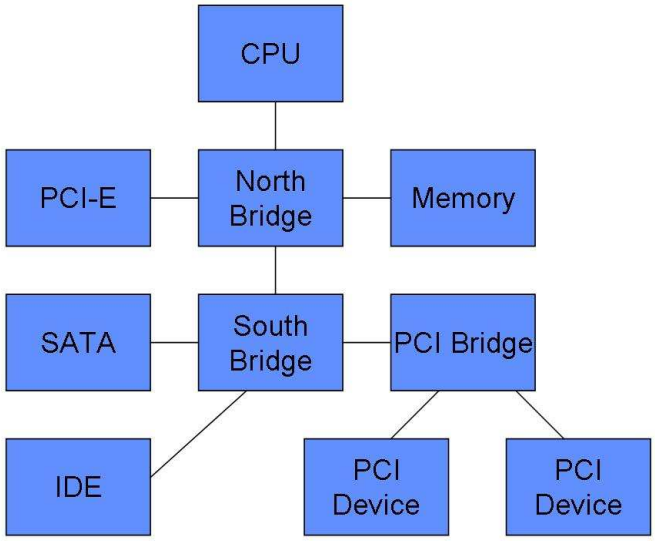

Figure 5.1: General Intel Architecture with PCI-Express

5.1.1.1 Northbridge. The northbridge of a system connects the processor(s), memory, and PCI Express slots. One of its key components is the memory controller. It also connects the rest of the peripherals attached to the system indirectly through the southbridge. Any communication between the processor(s) or memory and the rest of the system passes through the northbridge. This means that in order to gather *First-hand Information* from either the processor(s) or main mem-

ory monitors cannot wait until after this data has passed through the northbridge to monitor it.

5.1.1.2 Southbridge. Likewise, the southbridge connects I/O devices and other peripherals such as PCI devices back to the rest of the system. Although the potential impact on the system is less since these devices do not have as complete access to the core of the system, the potential exists for the southbridge to corrupt information being passed to or from any of these peripherals.

5.1.1.3 Processor. Above cited communications between the processor(s) and the rest of the system as partial reasoning for the northbridge as a critical monitoring point; however, due to the ability for malicious code to exist solely within cache, even monitoring the northbridge does not provide *First-hand Information* about the processor(s). Petroni et al. allude to the possibility of such an attack, but do not explain how such an attack would be accomplished [55]. Although no known instances of such an attack exist, the possibility for it does exist. If an attacker were to successfully inject a compact piece of code, which contained a polling loop to keep its utilization high, into the instruction stream it is potentially possible to keep this code in a processor's cache at all times. This code would remain their so long as it can maintain control of that processor [77]. To keep this attack undetected, another device, which does not participate in cache snooping, must repair the memory location that first allowed the attack. A peripheral can make this memory change without trigger a cache flush. This erases the evidence of the attack

from memory, hiding it's existence from current memory monitors. While flushing the cache will defeat this exploit, excessive flushing of the cache will create a significant performance degradation and must be balanced against the potential benefit. Even if only temporary, this theoretical attack demonstrates that at least until the cache is flushed, the northbridge does not receive accurate information.

In addition to this issue, the speed difference between the processor(s) and the northbridge provides windows of vulnerability even if lack of *First-hand Information* were not an issue. Furthermore, as Mott discussed [48], there are numerous pieces of information which can be useful for security purposes which are not currently made available off chip such as the program counter and instruction trace.

5.1.2 Differences in AMD Systems. The critical difference in AMD's architecture is the Hyper Transport (HT) Bus. HT changes the pathway for communications between the processor(s) and main memory, shifting northbridge functionality on-chip. The memory controller being on-chip allows for greater speed and integration. What is important to note here is that the need for on-chip monitors is even more prevalent within an AMD system since monitoring of the memory controller is crucial for maintaining *First-hand Information* of memory.

5.2 Security Backplane Architecture

Having explained the need for dedicated security hardware and presented the advantages of a security backplane system over other hardware options in Chapter III,

this section presents the details of the backplane architecture. This discussion begins at a high level, leveraging the discussion of critical monitoring and access points, to identify the general areas which this system will connect to a production system. Figure 5.2 presents this overall system design with red shadows representing locations where monitors and interaction points are located and red lines representing communication paths between the elements of the security backplane.

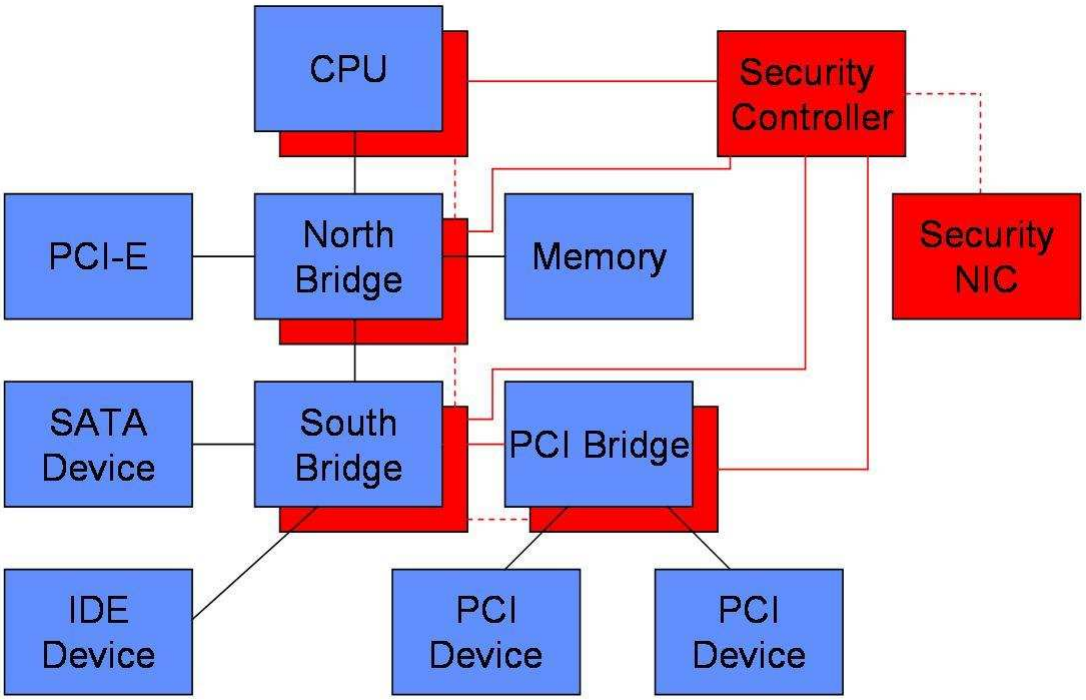

Figure 5.2: High-level Hardware Security Backplane Architecture Implementation Design

5.2.1 SHI(EL)DS Components and Monitors. Security monitors will be located at each of the critical interaction points discussed in Section 5.1 and connected to a main security controller. This controller will be responsible for policy decisions, security system communications, and implementing security updates. The monitoring

locations provide insight into numerous aspects of the system not previously monitored. This section presents a more detailed view of monitor placement within the system.

5.2.1.1 Security Controller. The security controller is intended as a main processor for the security backplane architecture. This processor extends the capabilities of Mott's shadow monitoring unit [48] to include a number of additional features. Mott's SMU is responsible for performing monitoring of memory, via the front-side bus, and monitoring of CPU state information exposed by his architecture. SHI(EL)DS security controller is designed to coordinate between the different security elements monitoring different system components. It also is responsible for managing information passed from the network backplane into the system and coordinating local security policies. This coordination of local security can include passing necessary information between components as well as enabling and disabling components to balance security needs with power consumption and performance degradation.

To enable monitoring of processes' memory, Mott presents a non-uniform memory access (NUMA). His NUMA architecture is designed to limit the visibility of the production processing unit into the SMU's memory space. This research proposes improving on the security of the memory space dedicated to running SMU software, by including separate dedicated memory for the security system. Mott's proposed NUMA architecture must rely on configurable data available during a system's boot process. This presents a possible avenue of attack into the SMU's exclusive memory.

By completely separating the security system's memory from the production system's memory, this avenue of attack is removed.

5.2.1.2 Processor State Monitors. Mott proposes exposing the program counter and the process identifier (PID) to aid in enforcement of execution policies and control peripheral access on a per process basis. These concepts are discussed in more depth in his thesis. SHI(EL)DS includes monitors to expose both of these signals. The monitors discussed in the following section give us the ability for physical memory introspection, however, the ability to examine virtual memory can provide valuable insight into a process' execution. The memory management unit, which is responsible for the translation from virtual to physical memory, is one of the crucial bottlenecks in a system. Due to its highly specialized and optimized design, modification of the MMU is impractical. However, we do not need to modify the MMU to determine the link between a virtual address and the corresponding physical address. Instead, a monitor that exposes the address sent from the processor to the MMU is provided. Since this monitor is completely passive it avoids negative impact to the MMU's performance. The only performance consideration for the capturing of this information is fanout created by the added wire and a simple buffer. Once the virtual memory address is exposed the security system must associate it with the physical address passed out of the MMU.

Mott presents work towards gaining access to virtual memory. He proposes two methods of achieving this: one that requires co-opting the MMU and another that

utilizes multiple MMUs on his SMU. Modification to the MMU presents significant risk to the performance of this system bottleneck. Due to this risk, this concept is designed to avoid interfering with the design of the MMU completely. The result is a focused, limited view into virtual memory that can be achieved without risk of compromising the performance of the production system. This provides a reduced capability to that provided by Mott's research, but also reduces impact to the production system. Though synchronizing the monitored virtual address with the monitored physical address for association will be a somewhat complex task, there is no increase to the bus traffic through this monitoring technique. If the security system has some knowledge of the operation of a production system process and its proper execution, it can detect attempted access to sections of code that are not meant to be executable.

Mott's multiple MMU concept shows promise for the capabilities it offers, but contains one significant drawback: high utilization of the memory bus. This bus is one of the bottlenecks in current computer architecture design. Though his research addresses this aspect, it is still an issue. Flooding this bus with memory access requests from multiple MMUs will present a significant risk of performance degradation. As production systems utilize more processors in their design the contention for this bus will increase. Any reduction in required access to the memory bus will benefit performance. Although the primary factor in motivating a completely separate memory for the security processor is increased security, this feature pays dividends in this situation also. Memory access by the security controller for its own code utilizes a separate bus, relieving some of the congestion present on the production system's

memory bus. This reduces the impact of the security system on the performance of the production system.

Note that exposing new information on the processor will require dedicated pins to pass the information off-chip to a dedicated security bus. Alternatively the security controller processor could be integrated into the main chip. This would still require dedicated pins to connect to a security bus passing information back from other monitored components in the system. Figure 5.3 displays these monitors as well as the monitor from the following section that provides access to physical memory. The virtual and physical addresses that are gathered by the two monitors are associated by the security controller providing focused visibility into virtual memory space. The dotted line denotes what is included on the main chip in current AMD systems.

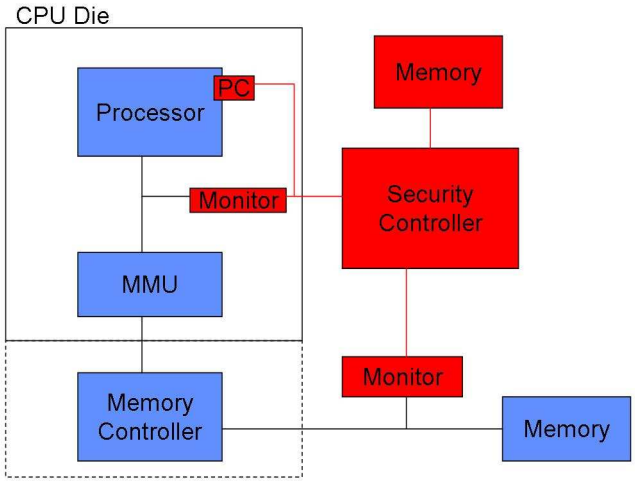

Figure 5.3: Processor primitives, virtual address, and physical address are all monitored and fed to the security controller.

5.2.1.3 Northbridge Monitors. The northbridge connects a number of components of the system together. It passes information from the processor, main

memory, the southbridge, and key performance critical devices such as the graphics processing unit (GPU). One of the critical components of the northbridge is the memory controller. The memory controller bridges the processor(s), main memory, and the rest of the system. This presents us with a question of where precisely to monitor the memory controller and how to connect to it. Monitors can be placed inline between the memory controller and memory to provide *Immediate Information* of what is being transferred to and from the memory controller. This monitor can also be placed to snoop the memory bus, achieving *First-hand Information*. Since the memory bus is not a shared bus architecture, *First-hand Information* provides equivalent reliability to *Immediate Information*.

Another important monitor to place on the northbridge is one that has visibility into the memory-mapped I/O (MMI/O) registers. In AMD®systems these registers are responsible for mapping memory requests to I/O space. Since the processor only accesses these registers when specifically intending I/O interaction and peripherals base their lookup purely on address, this provides an opening that can provide differing interaction for the processor and peripherals [63]. This is the corruption of the northbridge that Rutkowska exploits to defeat hardware-based memory acquisition solutions such as CoPilot [55]. By monitoring the values of these registers this monitor will have visibility into attacks that attempt to cover a portion of main memory from peripheral access.

5.2.1.4 Southbridge Monitors and Interaction Points. Since the southbridge is responsible for the bulk of I/O communication it is an ideal place to add isolating points that are capable of shutting off peripherals. Section 5.3.4 describes such a capability in more detail, using firewire as an example. Monitors on the southbridge observe the memory-mapping registers to ensure proper routing of information to and from I/O components.

5.2.2 Security Backplane Communication.

5.2.2.1 Internal. To allow separate elements of this security backplane to operate together, the system requires dedicated buses to transmit data between them. This addition, though logically easy to accomplish since there are plenty of bus standards available to choose from, will however be one of the most intrusive elements in a commercial product. It almost necessitates the need for motherboards specifically designed to accommodate this system for different elements of the security backplane to communicate with each other. Although the potential does exist to create plug-in[1] devices which reside between the different devices and their motherboard connection which contain their own ports for cables to connect them, this will quickly clutter the system and potentially lead to unacceptable slow down in some instances. This plug-in methodology does provide a potential route for implementing a complete system prototype more quickly than a system which requires extensive modification to a

[1]To include solder-on devices as well as explicit plug-in devices.

standard motherboard. Details for a proper implementation of a system prototype are discussed in 7.2.1.1.

5.2.2.2 External. Communication between different system's security backplane is accomplished through the use of a dedicated security network interface card (SNIC). This SNIC extends the single system security backplane into a backplane running through an entire security network, such as for a server farm. The security backplanes communicate with each other information such as the makeup of network traffic to the production system and that a specific system is assumed compromised. This provides a number of increases to the inherent security of a server farm. A system which becomes compromised normally becomes an avenue of attack into the entire network, but by connecting the network through a separate security backplane network a system which is identified to be compromised can be isolated from the network by instructing each system to reject packets from the compromised system.

Another aspect of external communication within the SHI(EL)DS architecture is the ability to share information of attacks between different systems within the network being protected and use this aggregate knowledge to better understand the extent and goals of attacks. While a single system network IDS can gather only limited information about what is transpiring on the network landscape, a network interface based IDS is able to grab a broader and more useful picture of attempted attacks [13]. The backplane architecture expands upon this ability by being able to

leverage communication between *all* protected systems within a network to increase the level of detail that can be gathered on attempted compromise.

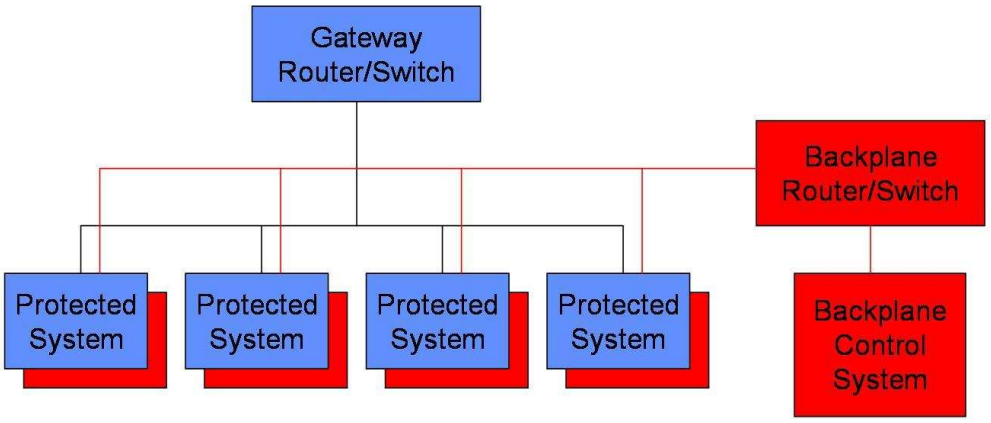

Figure 5.4: Depiction of a separate Security Network connecting a number of systems all equipped with the Hardware-based Security Backplane

An alternate method of communicating with external sources in a more remote environment, where we might not have full control over the network a system is on, is through the use of specialized hardware placed on the NIC. This hardware would be responsible for interpreting specific commands meant for the security backplane. These commands would be securely sent through the use of a encrypted protocol to transfer data from a remote source to the system. This concept can aide in developing a remote Cybercraft deployment platform, discussed further in Chapter VI. When the NIC is modified to provide a special communication pathway to the security backplane, the potential exists for relatively secure communication being passed to the security backplane from a remote source. The level of this security is critically dependent upon the security of the encrypted data it receives. This alternate method is crucial in any environment where we do not have the physical security or capability

to run a secondary network connection to the system. While a system without physical security is more vulnerable than a physically secured system, this security backplane still improves upon security. This improvement is due to the only interaction with the security backplane being through the network connection. By requiring knowledge of both the update protocols and the specific encryption methods and keys used to protect communication to the system, the main avenue of attack into the security system through the NIC is defined.

5.3 Capabilities

5.3.1 Directly Monitor Memory. One of the most core, important capabilities that a security system must have is the ability to monitor *First-hand Information* from main memory. Since current architectures allow memory access from peripheral devices, avenues of attack exist to compromise a system without running code on the processor. This is accomplished through peripherals uses direct memory access (DMA). By adding hardware that directly monitors the memory controller, be it on-die or on the northbridge chipset, this system guarantees that the monitor feeds accurate information about what is stored in and requested from memory to the security system. Figure 5.5 shows a notional example of Rutkowska's attempted subversion of hardware on a system protected by the SHI(EL)DS architecture. Since the monitor accesses the communications between main memory and the memory controller, regardless of where information is passed by the memory controller, the monitor will accurately see the information being passed out of memory *and* know

where the memory controller is directing the information. This is shown in Figure 5.5 by the security monitor covering the northbridge, depicted in red.

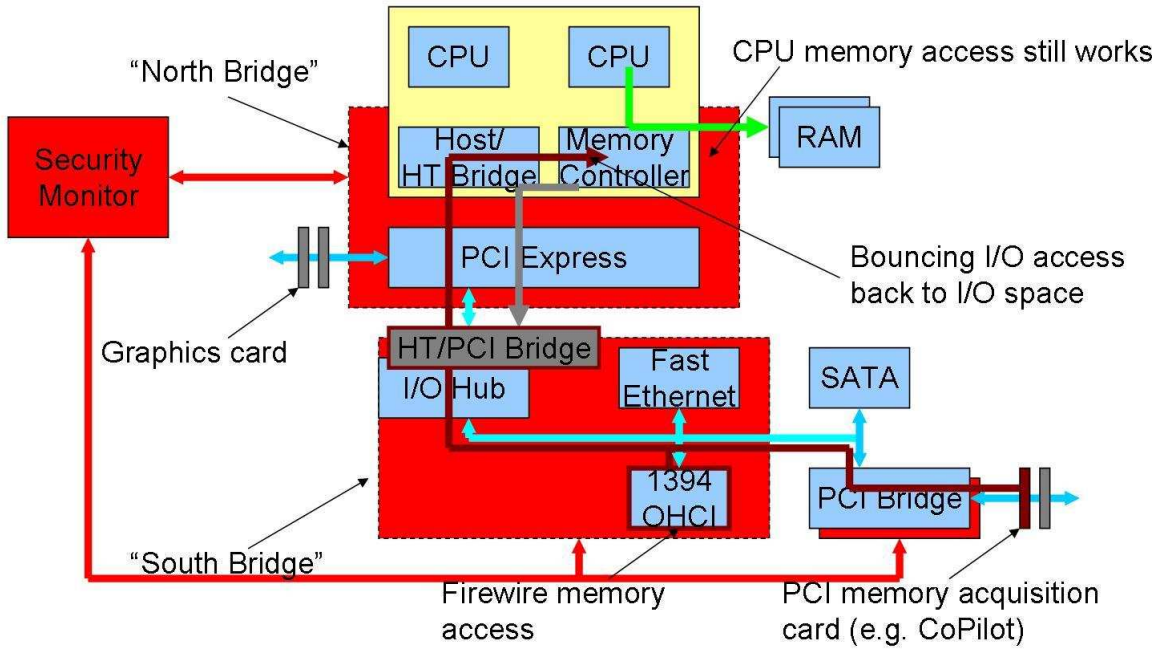

Figure 5.5: Hardware Security Backplane Response to Rutkowska's Hardware Based RAM Acquisition and PCI Bus Access

Figure 5.3 shows a more detailed view of a monitor gathering information directly from the memory bus. Since the monitor sits on the memory bus, it can provide real-time monitoring of memory. To ensure real-time monitoring the monitor and the bus connecting to the security controller must operate at least as fast as the memory bus transfer rate. Note that this ensures only real-time *monitoring*. Real-time *detection*, as defined by Kuperman in Section 2.2.3, can only be achieved if the security controller can analyze the monitored memory and detect malicious code before it compromises the system.

5.3.2 Towards Reliable Detection of Type III Malware. Rutkowska claims her *Type III Malware* is virtually undetectable since it does not modify the OS or VMM running on the system in any way. Therefore, even if both static and dynamic verifiers of the system were created, they would report a clean system. To overcome this ability to hide *Type III Malware*, the security monitor must be able to ensure that it has visibility into the entirety of the physical systems memory. This is a capability provided by the SHI(EL)DS architecture. Although this system provides this essential requirement for detecting *Type III Malware*, a complete verifier must be established at this level as well. Since verifiers to detect both *Type I Malware* and *Type II Malware* are not yet fully realized, a verifier that can detect *Type III Malware* is not yet a priority. However, more systems are being equipped with hardware assisted virtualization and utilizing VMMs. This will almost certainly lead to the use of *Type III Malware*. This architecture provides the necessary platform to develop a verifier to detect this type of malware by guaranteeing an accurate view of memory. A verifier built without the guarantee of an accurate view of physical memory cannot reliably identify *Type III Malware* since the verifier can be running inside of a hypervisor.

5.3.3 Ensure Peripheral Memory Request Matches Address Provided. The monitors depicted in Figure 5.5 also present another unique capability. By monitoring peripheral buses and the MMI/O registers on the northbridge this monitor can compare the memory address requested by a device, such as a PCI card, to the memory ranges mapped back towards I/O. If the comparator determines that a pe-

ripheral requests a memory location mapped back to I/O space it detects an attempt to mask memory. Security system can respond by repairing MMI/O registers affected or forcing peripheral request to actual memory.

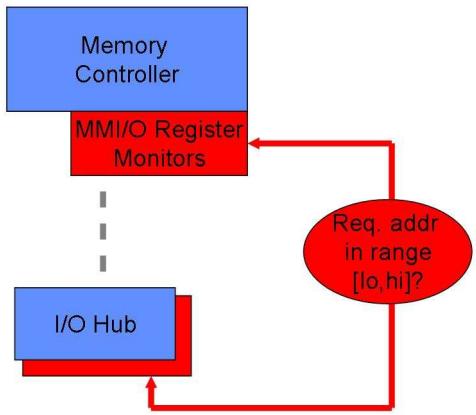

Figure 5.6: Comparator ensures that I/O memory request is not masked by MMI/O manipulation.

5.3.4 Isolation of Components and Systems When Compromise Detected.

By using simple fast transistor switches, controlled by the security system, each monitoring point can achieve the ability to shut off access on a temporary basis. An example where this capability provides added security is shutting off a specific peripheral that is attempting to compromise a system such as Rutkowska's avenue into the system for her exploit described in Section 2.1.3.1. Upon detection of Rutkowska's attack the security system sends a control signal to the transistors inline with each wire of the firewire port in question. Once the attack is detected, this control signal can be asserted within a clock cycle and the associated wire delay. The critical component in the speed of this isolation is the time to detect the attack.

When no attack have been detected from a specific peripheral the control signal remains de-asserted. A transistor with rise and fall times in the picosecond range will not impact Firewire 800, which has a period no shorter than 1.25 ns ($\frac{1}{800\ \text{Mbps}} = 1.25$ ns per bit).

This concept also extends to the security network backplane level. If a system in the network is recognized as compromised it can be isolated from the production network in a number of ways. Other systems in the network can be instructed to ignore all data being sent from this machine, protecting the compromise from spreading to other machines in the network. The control center, discussed in Section 5.3.7.1, can also inform the local backplane of the compromised system to respond locally to the threat, even if the local backplane had not previously detected a compromise on its system.

5.3.5 Verifying Integrity of Component Firmware and Boot Information.

Rootkits such as SubVirt, discussed in Section 4.2.5, gain control of a system before the OS has loaded and insert their VMM so that the OS runs inside of it. Research of this nature motivates the need to be able to verify the BIOS and firmware for system components has not been corrupted for an attack such as SubVirt. Having hardware monitors that are not modifiable by the production system located throughout the system provides a tool with which to perform this verification. The security backplane provides a high level of assurance that it cannot be modified by malicious code on the production system. This assurance can be leveraged during startup as a tool

to examine components firmware and boot information. A hash of proper code can verify that these devices have not been corrupted.

5.3.6 Combining Network Intrusion Detection and Host Intrusion Detection. Combining the knowledge gathered from the two main categories of IDS provides unique opportunities for aiding in the protection of the system. Each provide the other with information about the current state of the system which can be leveraged by the other to improve its security response.

5.3.6.1 Leverage Network Intrusion Detection Information for Host Intrusion Detection. Depending on the strictness of network ID filters, the number of potentially malicious packets of information passed into the system varies. The tradeoff between *True Positive* rates and *False Positive* rates has been reduced by work such as [25], but still remain a significant issue. One possible avenue of further reducing this tradeoff is to identify potentially malicious packets, which the system does not have a high confidence of being malicious, and pass them into the production system flagged as potentially malicious. The host-based ID effort can leverage this information to tighten or loosen the specifications defining proper and improper behavior in the system.

5.3.6.2 Leverage Host Intrusion Detection Information for Network Intrusion Detection. While the network ID can provide useful information to the host ID, the opposite is also true. If a host IDS recognizes that malicious code has

attempted to compromise some aspect of the production system, it can communicate this information to the network IDS, providing it concrete knowledge that malicious packets escaped detection. By having a time window of when a compromise occurred, these packets can be reexamined more in depth to find the specific offending packets. Without this knowledge, the network IDS has little feedback into the quality of its detection. Any increased feedback on the effectiveness of the network IDS provides added ability to tailor the detection process to be more accurate for the specific system it is attempting to protect.

5.3.7 Addition of Dedicated Systems to the Security Network. This networked backplane allows for systems to be connected through their regular NIC for purposes such as:

5.3.7.1 Control center for setting policies and updating the security backplane. Creating a security backplane that uses dedicated hardware to separate itself from the production system creates the need for an interface into the security system that does not rely on the production system. One method of accomplishing this is including a system attached to the network backplane to be used as a control system. This control system can be designed to allow a user interface into the state of the security system across the entire network. It provides a convenient, powerful interface to the security system without creating new avenues of attack into the system other than an attacker gaining physical access to the control center machine. This control center can be utilized to load the signature of a newly discovered rootkit into each

system in the SHI(EL)DS network. The local security system on each machine can then use it's direct memory monitoring capability to examine physical memory for this signature. This control center can also be used to upload new specifications or policies in response to outside events. For example, if the an Air Force base utilizing this system were to change from Infocon Alpha to Infocon Delta, this policy change could be fed through the control center to each SHI(EL)DS equipped system. Each system and backplane component can have a tailored response to this policy change. Systems acting as gateways to external networks can be updated to allow almost no network traffic, where systems not exposed can have less drastic responses.

5.3.7.2 Dedicated processing power for network based ID with data from each system. The processing power of additional computer systems attached to the network security backplane can be employed in a number of ways. By each system's security backplane transmitting suspicious packets to a central source, the information can be examined by a more robust network IDS located there, which has the added knowledge of network traffic to each individual system in the security backplane network. This can be done to create a central network IDS to govern internet access or combined as a second layer to work on top of more typical IDS. One advantage this has over a typical network IDS is the added security the network level systems have. This extra security is provided by the systems not being accessible by external Internet sources, except through a compromise of the local SHI(EL)DS architecture. Network IDS must balance their *True Positive* and *False Positive* rates when detect-

ing potentially malicious packets and perform the entirety of their decision process fast enough to block what they identify as malicious (within Kuperman's *Real-time Detection*). By creating a second-level to the IDS, this demanding timetable can be relaxed, allowing the second-level system to perform significantly more in depth analysis of the potentially malicious packets. This relaxation is created through a restrictive first-level classifier that is biased towards producing positives. This helps to increase the *True Positive* rate, but also incurs a higher *False Positive* rate. This first-level classifier still requires a quick response time. However, these potentially malicious packets can be sent to the central processing location where this more in-depth analysis will occur. Packets identified here as non-malicious can be transmitted back to their respective production system. Although there is significant delay added to the time for these packets, the automated second-level identification of *False Positives* remains many orders of magnitude faster than passing potentially malicious packets to an operator such as a network administrator.

Appendix A contains an extension of the jREMISA research that provides a proof-of-concept for this two-layered approach. This proof-of-concept takes a standard jREMISA learning run with a comparatively high *True Positive* and *False Positive* rate and flags each potentially malicious packet to be fed into the second-level. At this second-level the jREMISA learning process is run to create detection antibodies that focus on the differences between only those packets identified as potentially malicious at the first-level. Appendix A presents a short comparison between a standard jREMISA solution and this two-layered solution using jREMISA at both levels. While

this proof-of-concept provides only a small improvement, quantified in Appendix A, it achieves this without any increase to the allowable processing time. Using a dedicated processor attached to the network security backplane would allow for not only increased processing time, but also significantly more storage for antibody definitions. This would help to increase the resolution of detection. The network SHI(EL)DS architecture provides a significant storage increase due to the ability to dedicate an entire system, or cluster of systems if necessary, to provide this increase. Such a system could easily achieve Terabytes of storage, though the more this increases the greater the computational time of the second-level will need to be.

5.3.7.3 Storage location with data collection for forensics. Although this capability is beyond the scope of this research, the networked security backplane provides valuable options to forensic capabilities. Tuting and Williams [74] explores this capability, discussed in more detail in Section 7.2.3. The key aspect of this capability is the ability to leverage a large amount of dedicated storage to record the desired forensic data.

5.4 SHI(EL)DS' Weaknesses

Despite the significant improvement offered by this architecture, it is not a perfect solution. This section presents a number of negative aspects of this system and vulnerabilities not addressed by the hardware-based security solution.

5.4.1 Physical Security. While one of the key improvements this security architecture has provided is towards the security of the *security system*, it is important to note that the architecture is not an impenetrable security system. One clear example of this is the physical security of a system equipped with SHI(EL)DS. As with any computer system, if an adversary can physically manipulate the system, security becomes all but impossible to maintain [17].

5.4.2 Design and Implementation Cost. Although this design focuses on minimizing *changes* to the production system, almost every element of this system requires additional hardware. Offsetting the high development cost associated with this implementation, is the minimal impact it will have on the production system. Eliminating redesign of production hardware and the associated ISA reduces the integration costs. This hardware will be integrated throughout the entire motherboard in production systems, with the most notable addition being a dedicated security processor and memory. Much of this new hardware will be monitors which must be designed to successfully capture the targeted information accurately and securely. These monitors and the dedicated security processor will incur a significant cost for their design and development. This cost presents a significant obstacle to overcome in terms of developing the SHI(EL)DS architecture, but these costs are offset by the security gains of the proposed solution presented in this chapter. For security critical systems and networks, additional costs for improved security is a reasonable expenditure. A SHI(EL)DS equipped system would present a costs estimate less than double

that of an unprotected system, based purely off component costs. This rough estimate assumes to key pieces of information to cap the cost estimate: not every single component is monitored, and monitors are comparatively simple pieces of hardware to what they are monitoring.

VI. Integrating SHI(EL)DS Into the Cybercraft Initiative

This chapter presents the Air Force's Cybercraft initiative and show how the SHI(EL)DS research integrates into it. It begins by describing the basic ideas behind Cybercraft and key ideas and capabilities relating to Cybercraft. Once the basics of this initiative are established, the benefits of using SHI(EL)DS in Cybercraft are shown and key capabilities provided are explored. This chapter looks at both the general concepts and more specific details in terms of how this research improves Cybercraft.

6.1 Cybercraft

The overarching goal of the Cybercraft initiative is to create a trusted deployment platform for blue information systems [69]. Current research focuses on deploying Cybercraft to internet protocol (IP) administration networks, with the intention of eventually covering the entire blue cyber infrastructure. Cybercraft provides for the deployment of defensive capabilities to the cyber infrastructure through a *trusted* platform.

6.1.1 Cybercraft Architecture Objectives. In creating the basic architecture specifications, four key objectives are identified [7]. These objectives influence the design direction in different ways. The second and third objectives pertain mostly to the intercommunication architecture between systems. The first and fourth each influence the architectural requirements of the Cybercraft specification significantly.

Simple Must have a consistent design

Scalable Support operations on more than one million systems

Reliable Cannot allow single points of failure

Provable Provably secure operation of the Cybercraft is paramount

6.1.2 Cybercraft Problem Domains. AFCYBER Mission Roles and Responsibilities described in [75] are mapped to Cybercraft problem domains in an attempt to bridge the transition between operational and real world requirements. The Cybercraft problem domains are listed here with some basic descriptions [69, 21].

C3 Protocols and Architecture Command, control, and communication (C3) protocols need to support more than one million nodes with a secure architecture.

Map and Mission Context A key role of Cybercraft is distributed information retrieval.

Environment Description To understand the environment that Cybercraft are deployed in the system must merge sensor information into an understanding of the state of the system

Formal Model and Policy Formally modeling the Cybercraft and establishing policies will help ensure that the system operates in the manner intended by commanders.

Payload Interfaces These interfaces need to be dynamically alterable. This allows for updated solutions to be reliably uploaded to the Cybercraft platform.

Self-Protection Guarantee Creating a Cybercraft deployment platform which can provide a high level of security to itself is essential for continued trusted operations of the Cybercraft infrastructure.

6.1.3 Main Cybercraft Components. The Cybercraft concept at the individual systems level is composed of two main ideas: the Cybercraft deployment platform and payloads which are loaded onto them [21]. This section presents the basic description of each here and attempts to draw clear lines between the roles of each in a systems defense.

Cybercraft Deployment Platform Trusted mechanism residing on protected systems, responsible for providing a framework to gathering data, interacting with the system, and communicating with the entire Cybercraft infrastructure. The key design focus is to provide the necessary capabilities, in a trusted platform.

Payloads Specific sensors, decision engines, and effectors which are uploaded to the Cybercraft deployment platform.

This research fits well into the goals and requirements for the Cybercraft deployment platform. The following section describes in detail the aspects of this system that provide a basis for developing the Cybercraft deployment platform. It also ties this development back to the Cybercraft problem domain discussed in Section 6.1.2

6.2 Mapping Cybercraft Deployment Platform Requirements to SHI(EL)DS Capabilities

6.2.1 Trusting the Cybercraft. One of the most critical issues of Cybercraft is the ability to trust the platform and it's payload to operate correctly. As seen in Section 6.1.2, the ability to guarantee self-protection is one of the core Cybercraft problem domains. Two key components to achieving this is establishing a *root of trust* and reducing the avenues of attack to a defendable quantity. The SHI(EL)DS architecture makes significant progress towards both of these requirements.

6.2.1.1 Root of Trust for Cybercraft Deployment Platform. The SHI(EL)DS research provides the potential for meeting all three *roots of trust* as defined by the TCG [24] in Section 2.3.2.

Root of Trust for Measurement (RTM) As this research has developed, the only way to trust measurements about the production system is to achieve at least *First-hand Information* for anything that is being measured. This system is designed to gather at worst *First-hand Information* from everything that it monitors.

Root of Trust for Storage (RTS) Significant work has gone into attempting to create trustworthy storage via both maintaining higher levels of privilege for the security systems storage and through encrypting the data being stored. By creating a physically separate storage for the security system, it eliminates *all* direct access to the security systems storage.

Root of Trust for Reporting (RTR) The separate security network presented in this research provides an avenue for trusted reporting. By not relying on the production system to transmit messages this system presents a more secure communication pathway.

By proposing a system that can provide each of these *roots of trust* this research enables a high level of confidence in this security architecture. Since this architecture does not allow for the production system to modify the *code* of the security system, barring an attack by someone with physical access to the system to compromise the security system software, when the system is initiated it is in a trustworthy state.

6.2.1.2 Reducing Avenues of Attack. The SHI(EL)DS architecture is designed to allow only constrained hardware-based communication pathways between the production and security systems. This limiting of the communication reduces the main avenues of attack on the security system to a small set of mainly passive monitors. These monitors can be designed with security as their primary focus, not performance, since they will have little to no effect on the production system, unless responding to an attack. Many current security solutions utilize software communication from the kernel and share main memory. Both of these aspects present avenues of attack upon the security system.

6.2.2 Accurate Information Retrieval. Leveraging an understanding of the need for *First-hand Information* to guarantee accurate information retrieval by the security system, the SHI(EL)DS architecture is in a unique position to ensure that the

information gathered is indeed representative of the state of the monitored system. As we become aware of new pieces of information which will benefit the Cybercraft mission, new modules can be designed to fit into the SHI(EL)DS architecture that achieve at least *First-hand Information* about the targeted mechanism. Another key aspect of the information retrieval provided by this security system is that it can continue to report information on the state of the production system after that production system has been compromised. Since there is a separate backplane architecture, an attack must also compromise it to reliably achieve a stealthy attack on the overall system. This will allow for insight into the malicious activities being performed by the compromised system to be leveraged by the entire Cybercraft infrastructure.

6.2.3 Merging Gathered Information. The inclusion of a dedicated security processor in the SHI(EL)DS architecture allows for a secure location to analyze data on the state of the production system. From there the security processor can both react to potential threats it sees on the local machine and transmit secure communications back to a higher level system in the Cybercraft communications architecture. If this information must be passed over normal Internet connections, strong encryption must be employed to secure it as best as possible. Even if the information is passed solely over the security network backplane, encryption should still be used to help protect against the possible physical breach threat. This data can be compiled from a multitude of Cybercraft deployment platforms to provide a understanding of

trends in systems usage and provide additional knowledge to make decisions about the appropriate response of individual Cybercraft.

VII. Conclusions

We conclude this work be reviewing the key findings of this research and presenting a number of areas for future work which are illuminated by it.

7.1 Conclusions

7.1.1 Trustworthiness of Information. By developing this axis of classification, key insights into the development of security systems are gained. Design of new security systems will have a clear understanding of what is required to guarantee accurate information about the production system and what they sacrifice by not achieving at least *First-hand Information*.

7.1.2 Necessity of Dedicated Hardware for Security. This research presented a number of key advantages offered by properly designed security hardware and explored what is necessary for proper design. This research into the design of hardware is critical, since improperly designed hardware leaves significant vulnerabilities in the system. Each of the following abilities is gained only through the use of dedicated hardware or is significantly enhanced by its use.

Reduced Avenues of Attack Separating the security system into dedicated hardware allows for defining the exact connections between the security and productions systems.

Ability to Gather Trustworthy Information This new axis defined illuminates the need for security solutions which can provide *First-hand Information*. Only hardware-based solutions can accomplish this.

Additional/Different Information Available Hardware dedicated to monitoring information not normally exposed by the processor, such as Mott's research, provides previously unavailable information to be leveraged for security purposes. An example of this include the value of the program counter register.

Timeliness of Detection Network ID and processor monitoring are two scenarios where the speed of dedicated hardware can greatly enhance the security capabilities.

Physical Response Dedicated hardware can provide added physical security for remote computer systems by providing a destructive response when compromise is detected.

7.1.3 Requirements for Effective Hardware Design. By defining a list of requirements for hardware-based security system design this research provides vital insight into the design of future security systems. These requirements, relisted here, each provide unique aspects of security to potential solutions. The first four design elements listed below form requirements for any partial or complete security solution to be effective and secure. The final design element is required for producing complete security solutions.

- First-hand information of all monitored information

- Dedicated monitors for parallel, concurrent monitoring
- Explicit hardware communication between the production and security systems
- Dedicated storage of security code and data
- Dedicated security processor for controlling and coordinating the security mechanisms

7.1.4 Modular, Hardware-based Security Framework. Using the insight from the above contributions, this work developed key aspects of an extensible security solution. This hardware-based security backplane, SHI(EL)DS, provides a modular framework which can be extended both with new local security mechanisms and into a network security backplane. It provides a comprehensive system approach to gather unique, accurate information from system components and make the security system itself more secure. It is designed to adhere to each of the requirements reiterated in Section 7.1.3 and provide significant advantages to each of the benefits discussed in Section 7.1.2.

7.1.5 Cybercraft Deployment Platform Development. The Cybercraft initiative represents a new focus on more comprehensive security solutions throughout the entire cyber infrastructure. The SHI(EL)DS architecture provides a basis for the Cybercraft Deployment Platform and provides significant progress to many of the essential requirements of Cybercraft. SHI(EL)DS' improvement towards the security and trust of the Cybercraft Deployment Platform is provided by the hardware separation which reduces the avenues of attack on the security system to a limited and

well understood set of hardware communication pathways. If these pathways can be proven secure, it is possible to prove the entire security system is also secure from an attack that originates on the production system. To accomplish this one must also prove that the understood hardware communication pathways are the only avenues of attack from the production system to the security system. The SHI(EL)DS architecture also involves hardware primitives designed to gather information not previously available to the system. This information helps to provide the Cybercraft payloads with a greater depth of information to apply sensors, decision engines, and effectors to.

7.2 Future Work

While this research presents a number of key contributions to the field of computer security and the need for hardware in this field, it also opens up numerous avenues of research to press forward with. This section describes some of these areas of future work here.

7.2.1 Security Backplane Prototypes. This research has developed a system-wide hardware-based security backplane architecture. Although this system provides the potential to become the base of a complete security solution, prototyping the SHI(EL)DS architecture is a massive undertaking. This section presents a few suggestions of the first steps towards prototyping this system and demonstrating key abilities that this system provides.

7.2.1.1 Local Security Backplane Prototype. The primary system elements which need to be developed are the dedicated security processor and memory, key monitoring devices such as the memory controller, and the dedicated security communication pathways. By beginning with the security processor and communication pathways this achitecture develops a core system which can be built upon to add features. A first prototype would involve mechanisms such as a SRAS discussed in Section 4.3.4.2. Another important monitor to be included in a SHI(EL)DS prototype is one which monitors the memory controller. This key location begins to spread the focus of computer security out from the current two focuses: processor ID and network ID. While these two areas are arguably the most important areas to secure, Rutkowska's work [63] demonstrates that they are not the only avenues of attack to compromise a system.

7.2.1.2 Security Backplane Communication. Another important area of future research is exploring ways to combine information from different mechanisms within the security backplane architecture. One straightforward example of this mentioned in this research is when the local security system recognizes the production system is compromised. It can either cut its network access on its own or communicate with other networked backplanes to ensure they block communications from the compromised production system combining elements of host based IDS and network IDS. While this example shows obvious benefit, other communications and coordination between mechanisms of the security solution can provide the potential

for increased flexibility and accuracy of response. A couple possible examples of this communication include:

- Allowing suspicious, but not known malicious, packets onto the system and notifying other security mechanisms to suspect compromise more readily.
- Notifying the network IDS portions of all systems when a system is known to be compromised to help update and improve their identification process. This idea is discussed in slightly more detail below.
- Isolating system components which are likely to be corrupted and attempting to continue operating at a reduced capacity.

7.2.1.3 Networked Security Backplane Prototype. While expanding the SHI(EL)DS concept into a network security backplane requires a local prototype to be established first, a proof-of-concept can be realized using standard hardware much more quickly. A simple scenario can be modeled through simulating a compromise of a production system and providing notification to a central network security system, which leverages the recorded network traffic of a known compromised system to more accurately identify malicious packets. By leveraging this capability this architecture can gain an ability to shape the network ID response through this out-of-band analysis. This ability is similar to the learning phase for artificial immune systems. Having this feedback improves the adaptability of this security system. This improvement comes from being able to leverage the detection of malicious packets on one system to update the future detection criteria on another system. Modeling the production system

having both its internal and external network access blocked is also a realizable goal for a network security backplane prototype and can be used to minimize the impact of a single compromised system on the entire network's operation.

7.2.1.4 Incorporate SHI(EL)DS into Network Devices. Although this research is focused on a standard desktop or server computer architecture, the understanding gained in Chapter III can also be leveraged to adapt the SHI(EL)DS architecture to work on devices such as network switches, routers, and gateways. Extending the SHI(EL)DS architecture into these devices is the only way to achieve *First-hand Information* from networks' first line of defense. Research into this area can help protect networks from attacks that compromise these devices to provide internal network access and the ability to snoop internal communications. This research can also be tied into the other elements of SHI(EL)DS to provide an even more comprehensive security solution. Knowledge from each level of the security backplane can be combined to provide a more complete picture of what is happening on Air Force networks and enable more informed, detailed responses.

7.2.2 Combine SHI(EL)DS Architecture with Current Solutions. One of the most useful features of the SHI(EL)DS architecture is its ability to adapt other security concepts into the basic design. By leveraging a dedicated security processor, software-based solutions can be adapted to run without having to contend for production system resources. The central security processor with dedicated communications

pathways allows for an easily extensible backplane which can incorporate different hardware primitives and monitoring concepts.

7.2.2.1 Incorporating Network Hardware and Software Solutions. Network solutions such as Haag's jREMISA, discussed in Section 4.1.2, can be incorporated into the SHI(EL)DS architecture as software running on the security system. Once a SHI(EL)DS prototype is designed, software solutions such as jREMISA can be adapted to react to the information gathered by the hardware monitors.

7.2.2.2 Incorporating Hardware Primitives. Mott's work presents a number of hardware primitives which provide new information and new capabilities [48]. Each of these primitives presents a potential module to be added to the core SHI(EL)DS architecture. Research on the cost and benefits of each of these when incorporated into this system will provide useful direction into which security mechanisms to add to SHI(EL)DS first, to create a more complete security solution. This research will be aided by the understanding of the benefits and requirements of designing dedicated hardware for security.

7.2.2.3 Incorporating Other Hardware Solutions. Solutions such as the hardware-based network IDSs discussed in Section 4.3.3 present systems which can be adapted to operate within the bounds of the SHI(EL)DS architecture. Research into which of these systems provides the most beneficial and complete network protection and which will integrate with SHI(EL)DS most efficiently will provide a successful

leap forward in designing complete, hardware-based security solutions. These systems should be analyzed against the understanding of the capabilities hardware can provide. This work in understanding the need for dedicated hardware and the requirements for providing new or enhanced capabilities presents a path forward in analyzing proposed hardware solutions, both on their own and as an element to be incorporated into the SHI(EL)DS architecture.

7.2.3 Explore Network Backplane as Forensics Tool. By creating a security system with its own dedicated storage, this architecture improves the capabilities of computer forensics significantly. Tuting's thesis presents research into gathering accurate information of the production systems memory and analyzing it for forensic purposes. Significant progress can be made by melding this work in forensics into the SHI(EL)DS architecture. SHI(EL)DS provides this work with the potential for dedicated processing power to analyze the information gathered. It also provides a trusted avenue for receiving *First-hand Information* so it can ensure that it performs its forensic analysis on the actual state of the system.

7.2.4 Explore New Hardware Primitives to Add as SHI(EL)DS Modules. Work such as Mott's thesis present a number of different hardware primitives which provide unique information about a system that was not available without his proposed hardware changes [48]. With the understanding gained through this work in terms of what is required to yield security benefits from hardware, it provides a solid framework for exploring new locations within a system to insert hardware primitives

and monitor previously unavailable information. This work in understanding the need for hardware and developing the SHI(EL)DS architecture will also aid in understanding the cost/benefit analysis of different hardware primitives to add once they are discovered.

Appendix A. jREMISA Enhancements: Two-layer Network Intrusion Detection System

The goal of this software is to enhance the functionality of jREMISA [25] by tuning it to be extra aggressive in identifying *Non-self* packets and adding an additional automated decision maker (a second IDS) inline between the current AIS and the system administrators. By tuning the initial jREMISA algorithm to be extra aggressive, the focus of the secondary IDS changes from that of a standard IDS. This system now focuses on identifying those suspicious packets that are indeed *Self*, i.e. identifying each False Positive.[1] Many options are available to us for creating this second level of security. A second AIS or another form of Evolutionary Algorithm. Although the scale of the data has been reduced from the initial load to be examined the dataset is large enough to make classic deterministic search algorithms impractical.

With a primary focus on securing a system completely (at the risk of reduced functionality) the goals of ID can be focused specifically on the speed of detection, and the minimization of False Negatives. This focus will most likely result in a significantly increased False Positive rate as well. By creating a second layer to inspect purely the signals which have been blocked, timeliness becomes significantly less critical since the packet will not be allowed. This provides us the opportunity to leverage this extra time flexibility to create a significantly more complex second level detector which can provide more accurate identification and potentially allow falsely denied packages

[1]When examining the code base of jREMISA modified for these experiments it is important to note that the definition of False Positive and False Negative are reversed, i.e. a False Positive is a *Non-self* recognized as *Self* and likewise for the False Negative.

access once they have been cleared. Regardless of which type of algorithm is applied for the secondary system, in order to render it effective the selection criteria must be as orthogonal as possible. Careful tuning of parameters can help to accomplish this, as well as examining different portions of the packet in the initial jREMISA run and the secondary IDS run. The goal is to attempt to loosely and conservatively identify possible *Non-self* packets and then take a closer look at all of these packets, hoping to identify key aspects separating *Self* from *Non-self* which would not be apparent in or masked by the complete network data set.

One benefit gained through the use of a secondary IDS which is only examining those packets assumed to be *Non-self* is time. The reasoning for this is twofold: (1) since the first layer have already let through the major portion of *Self* packets is allows the majority of network traffic to continue unhindered through the system, only slowing down the suspicious packets and (2) since most current systems rely on a human in the loop such as a system administrator to provide this second level of defense, a substantial amount of slowdown would be required to present an automated secondary system which would not still be significantly faster than this human's identification of *Self* and *Non-self* packets. This increase in time presents us with the opportunity to implement a more detailed and accurate identification system by trading off speed performance. Significantly more involved systems can be implemented to leverage this idea while still providing performance far more expedient than the standard human in the loop. One more advanced option would be to offload all suspicious packets to a dedicated system which uses the entirety of its resources to reclassify packets as *Self*

or *Non-self* and return the former packets flagged to be allowed through the initial jREMISA system at the primary location.

A.1 Design of Experiments

As proof of concept these experiments run the jREMISA algorithm on a single day truth set, from the MIT Lincoln Laboratory 1999 Intrusion Detection Evaluation Data Sets [46], two times with varied parameters to simulate at least partially orthogonal selection parameters. When the attack data is run through the jREMISA algorithm with the first selection parameters all False and True Positives are recorded.[2] With the second selection parameters, only attack day data set packets which were flagged as *Non-self* through the system are run. By showing a resultant classification which is more accurate than either of the selection parameters run by themselves against the entire attack day data set, it demonstrates the viability of these enhancements.

A.2 Results

Once the negative selection phase of the jREMISA algorithm is completed on each day's data, the MOEA step is executed, recording all packets positively identified as *Non-self* regardless of this validity. This research achieves results relatively similar to that achieved by Haag. These results are shown in Table 1

[2]As defined in this document, not the jREMISA code base.

Attack Day	True Pos.	False Neg.	True Neg.	False Pos.
Monday	99.4346%	0.5653%	84.9497%	15.0503%
Tuesday	70.2127%	29.7873%	86.8922%	13.1078%
Wednesday	71.8897%	28.1103%	82.2047%	17.7953%
Thursday	94.4111%	5.5889%	77.3912%	22.6088%
Friday	99.8226%	0.1774%	72.7075%	27.2925%

Table 1: First Pass jREMISA MOEA results

Using the positively identified packets file, this software runs through the jREMISA algorithm again, focusing only on those packets which were positively identified for shaping of the MOEA. This research focuses on the Friday data, from the Lincoln Laboratory Dataset, since it provides the best coverage from the standard jREMISA first pass and therefore likely represents the most difficult to further improve. The results of this can be seen in Table 2. It is important to note that the percentages represented in this table are only in reference to the two specific categories from Friday's data in the previous table: the True and False Positives.

Attack Day	True Pos.	False Neg.	True Neg.	False Pos.
Friday	99.9613%	0.0388%	83.1607%	16.8393%

Table 2: Second Pass jREMISA MOEA Friday Results

Table 3 shows the updated overall percentage for the Friday data. This data shows a significantly reduced percentage of *Self* packets misidentified with a fairly small hit to the accuracy of identifying *Non-self*.

Attack Day	True Pos.	False Neg.	True Neg.	False Pos.
Friday	99.7840%	0.2160%	95.4041%	4.5959%

Table 3: Overall Enhanced jREMISA Friday Results

Although this decrease in the True Positive accuracy is contrary to arguably the most important goal there are a number of key points which mitigate this undesirable consequence. First and foremost, the code used to demonstrate this feasibility is optimized for a single pass on that data; with this two-pass approach the first approach can be specifically tailored to approach 100% accuracy on the True Positives, by deemphasizing the negative effect of a False Positive report. While this approach will present an increase in the number of False Positives, the second pass has been shown as a feasible method for reducing this increase. Secondly, since only the stochastic nature of the MOEA are used to create some level of orthogonality between the first and second pass filtering, this aspect has certainly not been exploited to its full potential. Finally, these feasibility experiments do not in any way leverage one of the most important advantages of this two pass system: a significant increase in the available processing time for the second pass. With data the presents the two-pass system as a viable method of further classifying network packets, this work shows that leveraging a two-pass concept in overall security system design can have a positive impact to overall accuracy.

Bibliography

1. Allen, J., A. Christie, W. Fithen, J. McHugh, J. Pickel, and E. Stoner. *State of the practice of intrusion detection technologies*. Technical Report CMU/SEI-99TR-028, 2000.

2. Anderson, James P. *Computer Security Technology Planning Study*. Technical Report ESD-TR-73-51, Electronic Systems Division, AFSC, October 1972.

3. Arora, Divya, Srivaths Ravi, Anand Raghunathan, and Niraj K. Jha. "Secure embedded processing through hardware-assisted run-time monitoring". *Design, Automation and Test in Europe, 2005. Proceedings*, 1:178–183, 2005.

4. Arora, Divya, Srivaths Ravi, Anand Raghunathan, and Niraj K. Jha. "Architectural Enhancements for Secure Embedded Processing". *NATO Security Through Science Series D - Information and Communication Security*, 2006.

5. Axelsson, Stefan. *Intrusion Detection Systems: A Survey and Taxonomy*. Technical report, Chalmers University of Technology, March 2000.

6. Bazaz, Anil and James D. Arthur. "Towards a Taxonomy of Vulnerabilities". *Hawaii International Conference on System Sciences*, 2007.

7. Bibighaus, David L. "Presentation on Cybercraft". Air Force Research Laboratory, Cyber Operations Branch, June 2006.

8. Bu, Long and John A. Chandy. "FPGA based network intrusion detection using content addressable memories". *Proceedings - 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, FCCM 2004*. IEEE Computer Society, Los Alamitos, CA 90720-1314, United States, April 2004.

9. Cahill, V., E. Gray, J.-M. Seigneur, C.D. Jensen, Yong Chen, B. Shand, N. Dimmock, A. Twigg, J. Bacon, C. English, W. Wagealla, S. Terzis, P. Nixon, G. Di Marzo Serugendo, C. Bryce, M. Carbone, K. Krukow, and M. Nielson. "Using trust for secure collaboration in uncertain environments". *Pervasive Computing, IEEE*, 2(3):52–61, July-September 2003.

10. Capra, Licia. "Engineering human trust in mobile system collaborations". *SIGSOFT Softw. Eng. Notes*, 29(6), 2004.

11. Carrier, Brian D. and Joe Grand. "A hardware-based memory acquisition procedure for digital investigations". *Digital Investigation,*, 1, February 2004.

12. CERT, (Computer Emerengy Response Team). "CERT Statistics: Vulnerability Remediation". 2007. `www.cert.org/stats/vulnerability_remediation.html`.

13. Clark, Chris, Wenke Lee, David Schimmel, Didier Contis, Mohamed Koné, and Ashley Thomas. "A Hardware Platform for Network Intrusion Detection and

Prevention". *Proceedings of the Third Workshop on Network Procesors and Applications (NP3)*. Madrid, Spain, February 2004.

14. Coello, Carlos A. and Nareli Cruz Cortés. "Solving Multiobjective Optimization Problems Using an Artificial Immune System". *Genetic Programming and Evolvable Machines*, 6(2), 2005.

15. Crosbie, Mark J. and Benjamin A. Kuperman. "A Building Block Approach to Intrusion Detection". *RAID '00: Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*. October 2001.

16. Edge, Kenneth S., Gary B. Lamont, and Richard A. Raines. "A retrovirus inspired algorithm for virus detection & optimization". *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*. 2006.

17. Fabian, Walter. "Physical access to computers: can your computer be trusted?" *Proceedings of the 29th Annual 1995 International Carnahan Conference on Security Technology*. IEEE, Piscataway, NJ, USA, Sanderstead, Engl, Oct 1995.

18. Foster, D. and V. Varadharajan. "Security and trust enhanced mobile agent based system design". *Information Technology and Applications, 2005. ICITA 2005. Third International Conference on*, 1, July 2005.

19. Garfinkel, Tal, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. "Terra: a virtual machine-based platform for trusted computing". *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. 2003.

20. Gillespie, Matt. *Intel® Virtualization Technology*. Technical report, Intel Corp., 2007.

21. Glumich, Sonja and Brian Kropa. "Cybercraft Briefing to JFCC". Air Force Research Lab, Information Directorate, November 2007.

22. Gonzalez, Jose M., Vern Paxson, and Nicholas Weaver. "Shunting: a hardware/software architecture for flexible, high-performance network intrusion prevention". *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*. 2007.

23. Gordon-Ross, Ann and Frank Vahid. "Frequent loop detection using efficient non-intrusive on-chip hardware". *CASES '03: Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems*. 2003.

24. Grawrock, David. "TCG Specification Architecture Overview, Revision 1.4", August 2007. `https://www.trustedcomputinggroup.org/groups/`.

25. Haag, Charles R., Gary B. Lamont, Paul D. Williams, and Gilbert L. Peterson. "An artificial immune system-inspired multiobjective evolutionary algorithm with application to the detection of distributed computer network intrusions". *GECCO '07: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*. 2007.

26. Hart, Samuel. *APHID: Anomoly Processor in Hardware for Intrusion Detection*. Master's thesis, Air Force Institute of Technology, March 2007.

27. Howorth, Roger. "Virtual Servers Pay Off". March 2003. `http://www.vnunet.com/itweek/news/2084897/virtual-servers-pay`.

28. Hutchings, B. L., R. Franklin, and D. Carver. "Assisting Network Intrusion Detection with Reconfigurable Hardware". *FCCM '02: 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 00, 2002.

29. Intel. "Introducing the 45nm Next-Generation Intel Core Microarchitecture". White Paper. `http://www.intel.com/technology/architecture-silicon/intel64/`.

30. Intel. *Intel Xeon Processor with 533 MHz Front Side Bus at 2 GHz to 3.20 GHz*. Technical report, February 2004. `http://www.intel.com/design/xeon/datashts/252135.htm`.

31. Intel. *Intel Trusted Execution Technology*. Technology overview, Intel Corp., 2007. `http://www.intel.com/technology/security/`.

32. Intel. *Intel Trusted Execution Technology*. Architectural overview, Intel Corp., 2007. `http://www.intel.com/technology/security/`.

33. Intel. *Intel Trusted Execution Technology*. Preliminary architectural specification, Intel Corp., August 2007. `http://www.intel.com/technology/security/`.

34. King, Samuel T., Peter M. Chen, Yi-Min Wang, Chad Verbowski, Helen J. Wang, and Jacob R. Lorch. "SubVirt: Implementing malware with virtual machines". *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, 2006.

35. Ko, C., M. Ruschitzka, and K. Levitt. "Execution monitoring of security-critical programs in distributed systems: a Specification-based approach". *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, 00, 1997.

36. Kuperman, Benjamin A. *A Categorization of Computer Security Monitoring Systems and the Impact on the Design of Audit Sources*. Ph.D. thesis, Purdue University, 2004.

37. Landwehr, Carl E., Alan R. Bull, John P. McDermott, and William S. Choi. "A taxonomy of computer program security flaws". *ACM Comput. Surv.*, 26(3), 1994.

38. Lee, Ruby B., David K. Karig, John P. McGreggor, and Zhijie Shi. "Enlisting Hardware Architecture to Thwart Malicious Code Injection". *Lecture Notes in Computer Science: Security in Pervasive Computing*. 2004.

39. Levine, J.; Owen H., J.; Grizzard. "A methodology to detect and characterize Kernel level rootkit exploits involving redirection of the system call table". *Information Assurance Workshop, 2004. Proceedings. Second IEEE International*, 107–125, April 2004.

40. Lindqvist, Ulf and Erland Jonsson. "How to Systematically Classify Computer Security Intrusions". *SP '97: Proceedings of the 1997 IEEE Symposium on Security and Privacy*, 154. IEEE Computer Society, Washington, DC, USA, 1997.

41. Lippmann, Richard, David Fried, Isaac Graf, Joshua Haines, Kristopher Kendall, David McClung, Dan Weber, Seth Webster, Dan Wyschogrod, Robert Cunningham, and Marc Zissman. "Evaluating Intrusion Detection Systems: The 1998 DARPA Off-line Intrusion Detection Evaluation". *Proceedings of the DARPA Information Survivability Conference and Exposition*. IEEE Computer Society Press, Los Alamitos, CA, 2000.

42. Loscocco, P., S. Smalley, P. Muckelbauer, R. Taylor, J. Turner, and J. Farrell. "The inevitability of failure: The flawed assumption of computer security in modern computing environments". *Proceedings of the 21st National Information Systems Security Conference*. October 1998.

43. McAfee Products, 2007. `http://us.mcafee.com/root/store.asp`.

44. McHugh, John. "Testing Intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory". *ACM Trans. Inf. Syst. Secur.*, 3(4):262–294, 2000. ISSN 1094-9224.

45. Medley, Douglas. *Virtualization Technology Applied to Rootkit Defense*. Master's thesis, Air Force Institute of Technology, March 2007.

46. MIT Lincoln Laboratory. "DARPA Instrusion Detection Evaluation Data Sets". `http://www.ll.mit.edu/IST/ideval/`.

47. Molina, Jesus and William Arbaugh. "Using Independent Auditors as Instrusion Detection Systems". *Information and Communications Security: 4th International Conference*, December 2003.

48. Mott, Stephen. *Exploring Hardware-Based Primitives to Enhance Parallel Security Monitoring in a Novel Computing Architecture*. Master's thesis, Air Force Institute of Technology, March 2007.

49. Mott, Stephen, Samuel Hart, David Montminy, Paul Williams, and Rusty Baldwin. "A Hardware-based Architecture to Support Flexible Real-Time Parallel Intrusion Detection". *Proc. 2007 IEEE International Conference on System of Systems Engineering*, 2007.

50. Nerenberg, Daniel. *A Study of Rootkit Stealth Techniques and Associated Detection Methods*. Master's thesis, Air Force Institute of Technology, March 2007.

51. Ning, Peng, Sushil Jajodia, and Xiaoyang Sean Wang. "Abstraction-based intrusion detection in distributed environments". *Information and System Security*, 4(4):407–452, 2001.

52. Norton Products, 2007. `http://www.symantec.com/norton/products/index.jsp`.

53. One, Aleph. "Smashing the Stack for Fun and Profit". *Phrack*, 1996. http://www.cs.wright.edu/ tkprasad/courses/cs781/alephOne.html.

54. Özdoganoglu, Hilmi, T. N. Vijaykumar, Carla E. Brodley, Benjamin A. Kuperman, and Ankit Jalote. "SmashGuard: A hardware solution to prevent security attacks on the function return Address". *IEEE Transactions on Computers*, 55, 2006.

55. Petroni, N. L., T. Fraser, J. Molina, and W. A. Arbaugh. "Copilot-a coprocessor-based kernel runtime integrity monitor". *Proceedings of the 13th USENIX Security Symposium*, 179–194, 2004.

56. Ragel, Roshan G., Sri Parameswaran, and Sayed Mohammad Kia. "Micro embedded monitoring for security in application specific instruction-set processors". *CASES '05: Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*. 2005.

57. Ray, Indrajit, Sudip Chakraborty, and Indrakshi Ray. "VTrust: A Trust Management System Based on a Vector Model of Trust". *Information Systems Security*, 91–105. 2005.

58. Robin, John Scott and Cynthia E. Irvine. "Analysis of the Intel Pentium's ability to support a secure virtual machine monitor". *SSYM'00: Proceedings of the 9th conference on USENIX Security Symposium*, 10–10. USENIX Association, Berkeley, CA, USA, 2000.

59. Robles, Sergi, Joan Mir, and Joan Borrell. "MARISM-A: An Architecture for Mobile Agents with Recursive Itinerary and Secure Migration". *2nd. IW on Security of Mobile Multiagent Systems*, 2002.

60. Rosenblum, Mendel and Tal Garfinkel. "Virtual Machine Monitors: Current Technology and Future Trends". *IEEE Computer*, volume 38, 39–47. IEEE Computer Society, Los Alamitos, CA, USA, May 2005.

61. Rushby, J. "Design and Verification of Secure Systems". *Proceedings of the 8th ACM Symposium on Operating Systems Principles (SOSP)*, volume 15.

62. Rutkowska, Joanna. "Introducing Stealth Malware Taxonomy". November 2006. http://invisiblethings.org/papers/malware-taxonomy.pdf.

63. Rutkowska, Joanna. "Beyond The CPU: Defeating Hardware Based RAM Acquisition", February 2007. http://invisiblethings.org/papers.html.

64. Rutkowska, Joanna. "Security Challenges in Virtualized Environments", October 2007. http://invisiblethings.org/papers.html.

65. Shi, Weidong, Hsien-Hsin S. Lee, Guofei Gu, Laura Falk, Trevor N. Mudge, and Mrinmoy Ghosh. "An Instrusion-Tolerant and Self-Recoverable Network Service Using a Security Enhanced Chip Multiprocessor". *ICAC'05: Second International Conference on Autonomic Computing*, 2005.

66. Snort, 2007. http://www.snort.org.

67. Song, Haoyu and John W. Lockwood. "Efficient packet classification for network intrusion detection using FPGA". *FPGA '05: Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays*. 2005.

68. Stakhanova, Natalia, Samik Basu, and Johnny Wong. *A Taxonomy of Intrusion Response Systems*. Technical Report 06-05, Department of Computer Science, Iowa State University, 2006.

69. Stevens, Michael. *Use of Trust Vectors in Support of the Cybercraft Initiative*. Master's thesis, Air Force Institute of Technology, March 2007.

70. Stevens, Michael and Paul D. Williams. "Use of Trust Vectors for CyberCraft and the Limits of Usable Data History for Trsut Vectors". *CISDA '07: Computational Intelligene in Security and Defense Applications*, 193–200. April 2007.

71. Summit, Xen and Elsie Wahlig. "AMD's Virtualization Technology, SVM". http://www.xen.org/files/xs0106_amd_virtualization.pdf.

72. Tang, Tiffany Y., Pinata Winoto, and Xiaolin Niu. "I-TRUST: investigating trust between users and agents in a multi-agent portfolio management system". *Electronic Commerce Research and Applications*, 302–314, 2003.

73. Tummala, Ashok Kumar and Parimal Patel. "Distributed IDS using reconfigurable hardware". *21st International Parallel and Distributed Processing Symposium, IPDPS 2007*. Institute of Electrical and Electronics Engineers Computer Society, Piscataway, NJ 08855-1331, United States, March 2007.

74. Tuting, Al Nath and Paul Williams. "Creating Hardware-based Primitives to Enhance Digital Forensics in a Novel Computing Architecture". *ICIW08: 3rd International Conference on Information Warfare and Security*. March 2008.

75. United States Air Force. "Air Force Doctrine Document: AFDD 2-X, Draft 3.1".

76. Vatsaas, Richard D. and David B. Erickson. "Anti-tamper Apparatus", August 2007. U.S. Patent 7256692.

77. Williams, Paul. "Ongoing Cache Exploit Research". Personal Conversation, February 2007.

78. Williams, Paul D. *Warthog: Towards a Computer Immune System for Detecting "Low and Slow" Information System Attacks*. Master's thesis, Air Force Institute of Technology, March 2001.

79. Williams, Paul D. *CuPIDS: Increasing Information System Security Through The Use of Dedicated Co-Processing*. Ph.D. thesis, Purdue University, August 2005.

80. Williams, Paul D., Kevin P. Anchor, John L. Bebo, Gregg H. Gunsch, and Gary D. Lamont. "CDIS: Towards a Computer Immune System for Detecting Network

Intrusions". *RAID '00: Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*. 2001.

81. Williams, Paul D. and Eugene H. Spafford. "CuPIDS: An exploration of highly focused, co-processor-based information system protection". *Computer Networks,*, 51, April 2007.

82. Yi, Sungwon, Byoung koo Kim, Jintae Oh, Jongsoo Jang, George Kesidis, and Chita R. Das. "Memory-efficient content filtering hardware for high-speed intrusion detection systems". *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*. 2007.

83. Zhang, Tao, Xiaotong Zhuang, Santosh Pande, and Wenke Lee. *Hardware Supported Anomaly Detection: Down to the Control Flow Level*. Technical Report GIT-CERCS-04-11, Center for Experimental Research in Computer System at Georgia Institute of Technology, 2004.

84. Zhang, Tao, Xiaotong Zhuang, Santosh Pande, and Wenke Lee. "Anomalous path detection with hardware support". *CASES '05: Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*. 2005.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
27–03–2008	Master's Thesis	August 2006 — March 2008

4. TITLE AND SUBTITLE

SHI(EL)DS:
A Novel Hardware-based Security Backplane
to Enhance Security with
Minimal Impact to System Operation

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Matthew G. Judge, Capt, USAF

5d. PROJECT NUMBER

JON: 07-152

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCE/ENG/08-07

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/SN (AT-SPI)
Attn: Robert W. Bennington
2241 Avionics Cir
WPAFB, 45433-7320
(937) 320-9068; robert.bennington@wpafb.af.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; Distribution Unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Computer security continues to increase in importance both in the commercial world and within the Air Force. Dedicated hardware for security purposes presents and enhances a number of security capabilities. Hardware enhances both the security of the *security system* and the quality and trustworthiness of the information being gathered by the security monitors. Hardware reduces avenues of attack on the security system and ensures the trustworthiness of information only through proper design and placement. Without careful system design, security hardware leaves itself vulnerable to many attacks that it is capable of defending against. Our SHI(EL)DS architecture combines these insights into a comprehensive, modular hardware security backplane architecture. This architecture provides many of the capabilities required by the Cybercraft deployment platform. Most importantly, it makes significant progress towards establishing a root of trust for this platform. Progressing the development of the Cybercraft initiative advances the capabilities of the Air Force's ability to operate in and defend cyberspace.

15. SUBJECT TERMS

Intrusion Detection, Hardware Security, Trustworthiness of Information, Memory Introspection, Cybercraft

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Major Paul Williams
U	U	U	UU	139	19b. TELEPHONE NUMBER *(include area code)* (937) 255–3636, ext 7253; paul.williams@afit.edu